TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Simulació de sistemes de partícules i generació de moviment orgànic amb Processing

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Sistemes de Telecomunicació

AUTOR: Joana Sansaloni Florit

DIRECTOR: Eulàlia Barrière Figueroa

DATA: 9 de juliol de 2009

Títol: Simulació de sistemes de partícules i generació de moviment orgànic amb Processing

Autor: Joana Sansaloni Florit

Director: Eulàlia Barrière Figueroa

Data: 9 de juliol de 2009

Resum

Aquest document conté l'anàlisi i resultats obtinguts en la “Simulació de sistemes de partícules i generació de moviment orgànic amb Processing”. Processing és un programa de codi obert basat en Java, que permet accedir ràpidament a la programació d'imatges, animacions i interaccions. La primera versió va sortir al 2002 i l'objectiu dels seus creadors, Ben Fry i Casey Reas, utilitzar la programació amb finalitats creatives.

Els objectius d'aquest treball són estudiar la generació de moviment orgànic i l'aplicació de les lleis físiques de la cinemàtica, l'estudi d'una llibreria que permeti la creació de sistemes de partícules i la programació d'una animació on s'utilitzin les eines adquirides.

La memòria ha estat dividida en 2 capítols: Moviment amb Processing i Sistemes de partícules. En el primer capítol es troba una introducció al programa Processing juntament amb l'explicació de com s'ha de programar un moviment i les opcions que es troben. Després s'analitza la programació dels diferents tipus de moviment tenint en compte les lleis de la cinemàtica i la programació de col·lisions. En el segon capítol es tracten els sistemes de partícules, les llibreries utilitzades i una explicació sobre la peça final. A més, hi ha un apartat dedicat a ProcessingBCN on vaig tenir la oportunitat de participar.

ProcessingBCN ha estat la primera trobada de Processing a Barcelona obert a totes les persones que treballin o tinguin algun interès en Processing. La idea neix de la necessitat de compartir experiència, coneixements i, simplement, ensenyar el que s'ha fet i veure el que fan els altres. Allà, vaig fer una presentació sobre moviment en Processing.

La part del projecte que ha donat més problemes ha estat la programació de col·lisions, sobretot quan es té més de dos objectes. Per últim, s'ha de tenir en compte que la creació d'una peça final és molt subjectiva, que depèn dels colors i el gust personal per tal que l'animació sigui atractiva.

Title: Simulació de sistemes de partícules i generació de moviment orgànic amb Processing

Author: Joana Sansaloni Florit

Director: Eulàlia Barrière Figueroa

Date: July, 9th 2009

Overview

This document contains the analysis and results obtained in the “Simulation of particle systems and generation of organic movement with Processing”. Processing is an open source software based in Java, oriented to the generation of images, animations and interactions in an easy way. The first version came out in 2002 and the objective of its creators, Fry and Casey Reas Well, is to apply programming to art.

The aims of this work are to study the generation of organic movement and the application of the physical laws of kinematics, the study of a library that allows the implementation of an animation in which the tools we learned are used.

This document is organized in 2 sections: Motion with Processing and Particle's systems. The first chapter is an introduction to the use of the program and the different options for the programming of motion. Then analyzing the programation of different types of motion applied to the laws of kinematics and programming collisions. The second chapter is about particle's systems, the libraries used and an explanation of the final sketch. Moreover, there is a section dedicated to ProcessingBCN where I had the opportunity to participate.

ProcessingBCN was the first meeting of Processing in Barcelona open to all persons who work or have an interest in Processing. The idea stems from the need to share experience and knowledge, or simply showing what one does and see what others do. There, I made a presentation on motion with Processing.

The most problematic part of the Project has been the programation of collisions, especially when it has more than two objects. Finally, it should be borne in mind that creating a final piece is very subjective, depending on the colors and the personal taste in order to do the animation is attractive.

ÍNDEX

INTRODUCCIÓ

Aquest treball és un projecte de programació gràfica amb el qual es pretén estudiar els models utilitzats per simular moviment realista tant d'objectes individuals com de sistemes de partícules, amb les seves interaccions i aplicant les lleis físiques de la cinemàtica.

Per portar-ho a terme, s'utilitzarà el programa Processing que és un llenguatge de programació basat en Java i de codi obert. Es tracta d'un entorn per a la programació en un context gràfic desenvolupat al MIT per Ben Fry i Casey Reas sota la direcció de John Maeda. La primera versió pública és de l'any 2002.

El mètode de treball que s'ha seguit és el següent:

- Familiaritzar-se amb el programa Processing.
- Estudiar les eines de generació de moviment i l'aplicació de les lleis de la cinemàtica.
- Estudiar una llibreria específica per a sistemes de partícules.
- Finalitzar el projecte amb la programació d'una animació (peça final), utilitzant les eines adquirides.

El treball s'estructura en dos capítols, les conclusions, la bibliografia i annexos.

El primer capítol tracta de com programar moviment amb el programa Processing. El primer apartat explica què és Processing, com funciona, a qui va dirigit, l'estructura bàsica d'un sketch i les primitives bàsiques. Després s'ha explicat què és el moviment i com es programen les animacions. Seguidament, s'han estudiat els diferents tipus de moviment aplicats a les lleis físiques i en temps discretitzat. Finalment, es parla sobre la programació de col·lisions i la complexitat que comporta.

El segon capítol fa referència a la programació de sistemes de partícules. S'explica la llibreria Physics que permet crear sistemes amb partícules i forces entre elles, regides per les lleis físiques. A més, d'altres llibreries utilitzades per fer la peça final. També es troba un apartat on s'explica la meva participació a ProcessingBCN, una trobada de gent que utilitza Processing i on vaig tenir la possibilitat de presentar el tema del moviment i la física aplicats en Processing. Aquest capítol, acaba amb una explicació de la peça.

Per tal de dominar el programa, als inicis es van fer moltes petites animacions i proves, fins arribar a l'anàlisi del moviment i les col·lisions. Les col·lisions han suposat més dificultat de la que imaginava, sobretot per la poca informació trobada sobre col·lisions de més de dos objectes.

Ambientalització: En tractar-se d'un projecte de programació, aquest treball no ha donat lloc a cap impacte ambiental.

CAPÍTOL 1. MOVIMENT AMB PROCESSING

1.1. Programació

Processing és un llenguatge de programació de codi obert basat en Java, que permet accedir ràpidament a la programació d'imatges, animacions i interaccions. És un projecte open-source iniciat per Casey Reas i Ben Fry, a l'Aesthetics and Computation Group Media Lab del MIT sota la direcció de John Maeda.

Un dels propòsits de Processing és actuar com una eina per aconseguir que no-programadors comencin a programar, gràcies a la gratificació que suposa el feedback visual instantani. També el que volien era desenvolupar una eina accessible, assequible i potent de codi obert, per això es va decidir que el software fos gratuït.

A l'estar basat en Java, pot agafar totes les seves funcionalitats, convertint Processing en una eina potent, sobretot per a projectes basats en la Web. La filosofia de programació és, com en Java, l'Orientació a Objecte. Actualment els encarregats de les millores i manteniment del projecte són un petit grup de voluntaris.

Tot i que Processing no és el primer llenguatge que atreu als artistes i dissenyadors, sí que és el primer que s'ha creat exclusivament per a l'art: Digital, Generatiu i Interactiu. L'art Digital fa referència a qualsevol tipus d'art on intervé un ordinador. Alguns exemples serien una animació, imatge, videojoc, etc. L'art Generatiu es genera, compon i es construeix de manera algorísmica per mitjà d'un procés matemàtic, mecànic o aleatori. I per últim es té l'art Interactiu que fa referència a qualsevol tipus d'art on l'espectador interactua amb l'obra. Per tant, Processing relaciona l'art amb la tecnologia.

Pel que fa a la interfície del programa és senzilla. Està composta per un editor de text per escriure el codi, una àrea de missatges, alguns botons i una sèrie de menús. Tot i que per a programadors experts aquesta interfície tan senzilla resulta massa bàsica, aquesta mateixa senzillesa és el que ha fet que persones amb poc o gens coneixement de programació puguin utilitzar el programa de manera bastant immediata.

En la figura següent es pot veure la finestra principal de Processing.

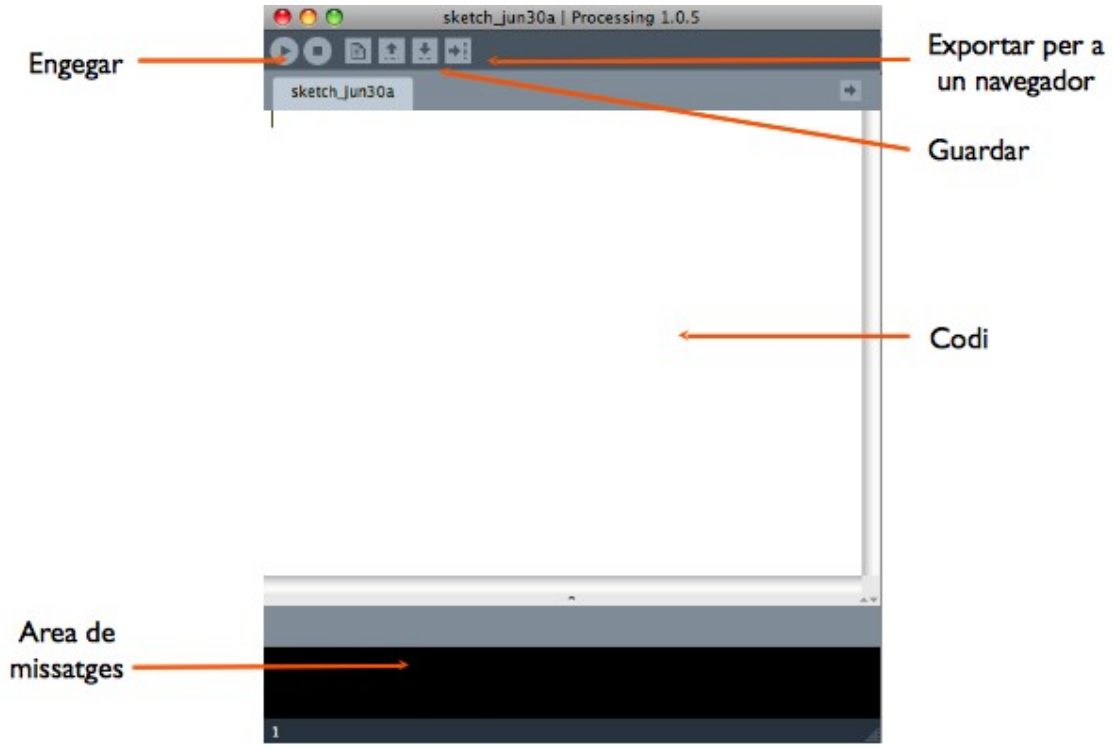


Fig. 1.1 Programa Processing.

Tots els projectes fets en Processing es diuen “sketches”, és a dir, esborranys. Cada dibuix té la seva pròpia carpeta amb el mateix nom que el programa principal. Dins aquesta carpeta també es guarden arxius multimèdia, llibreries o tipus de lletra que s'han utilitzat en el programa. És possible tenir diversos arxius de Processing en un sol sketch. Aquests serien funcions o classes que utilitza el programa principal.

La sintaxi del llenguatge és similar a la de Java, ja que n'és una extensió, però s'han afegit funcions específiques per el tractament d'imatges 2D i 3D, i per proporcionar interactivitat. Per donar una idea de quina és la filosofia de programació de Processing, donem a continuació una classificació de les primitives més bàsiques.

Paràmetres del dibuix

size(ample,alt);
Defineix la mida de la finestra on s'executarà l'sketch.

background(color);
Defineix el color del fons.

stroke(color);
Defineix el color de les línies.

strokeWeight(gruix);
Defineix el gruix de les línies.

noStroke();
Funció que fa que les línies no es dibuixin.

fill(color);
Defineix el color de l'interior de les figures.

noFill();
Funció que fa que les figures no s'omplin.

Variables del sistema

width: Amplada de la finestra.
height: Alçada de la finestra.

mouseX: coordenada x de la posició actual del ratolí.
mouseY: coordenada x de la posició actual del ratolí.

Colors

Per codificar els colors Processing disposa d'un tipus de dades propi, "color". La codificació més habitual és en RGB, encara que també admet el mode HSB. Per definir un color es pot fer de quatre maneres, segons que sigui blanc i negre o color i amb canal alfa (transparència) o sense.

blanc i negre = un nombre entre 0 i 255
blanc i negre, amb canal alfa = 2 nombres
RGB = 3 nombres
RGB amb canal alfa = 4 nombres

Primitives del dibuix

point(x,y);
Dibuixa un punt a la posició de coordenades (x,y) de la finestra.

line(x0,y0,xF,yF);
Dibuixa un segment del punt (x0,y0) al punt (xF,yF).

rect(x,y,dX,dY);
Dibuixa un rectangle amb el vèrtex superior esquerre al punt (x,y) i de dimensions dX, dY.

ellipse(x,y,dX,xY);
Dibuixa una el·lipse amb el centre al punt (x,y), d'amplada dX, i alçada dY. Si els eixos són iguals, es tracta d'una circumferència.

Interacció ratoli i teclat

En qualsevol sistema es disposa d'un tipus d'interacció bàsica, la que proporcionen el teclat i el ratolí. En Processing, es tenen variables i esdeveniments que permeten controlar aquesta interacció.

mousePressed();
Funció que hem de cridar per controlar què fa l'sketch en cas que es cliqui el ratolí.

mouseReleased();
Funció que hem de cridar per controlar què fa l'sketch en cas que es deixi anar el ratolí.

mouseMoved();
Funció que hem de cridar per controlar què fa l'sketch en cas que es mogui el ratolí.

mouseDragged();
Funció que hem de cridar per controlar què fa l'sketch en cas que s'arrossegui el ratolí.

keyPressed();
Funció que hem de cridar per controlar què fa l'sketch en cas que es pitgi una tecla.

keyReleased();
Funció que hem de cridar per controlar què fa l'sketch en cas que es deixi anar una tecla.

key: variable que guarda el valor de l'última tecla pitjada.

En aquest treball ens interessa la programació del moviment. El moviment és un fenomen físic en el qual els cossos d'un sistema experimenten un canvi de posició en l'espai respecte ells mateixos o amb un altre cos com a referència. Tot cos en moviment descriu una trajectòria.

En la programació d'animacions, una imatge és una matriu de píxels que s'actualitza cada vegada que es fa un refresc de pantalla. El temps és discret, per tant, s'ha de tenir una funció que s'executi cada unitat de temps i vagi canviant alguns atributs de l'objecte perquè sembli que l'objecte s'està movent. És a dir, es necessita un programa que s'executi contínuament per tal de poder actualitzar la matriu de píxels corresponent al dibuix. En Processing, aquest programa és la funció draw(). Abans de cridar-la es crida la funció setup() que serveix per inicialitzar les variables de l'sketch.

El draw() és un bucle infinit que s'executa un cert nombre de frames per segon. Per defecte, aquesta velocitat és de 60 fps i es pot canviar amb la funció frameRate(). El frameRate està limitat per la capacitat del nostre processador, ja que en el cas de posar un frameRate molt gran o si l'animació requereix molts càlculs, pot succeir que el processador no sigui capaç de suportar aquesta càrrega i per tant, no arribarà al frameRate desitjat.

El diagrama de flux bàsic d'una animació en Processing és el següent:

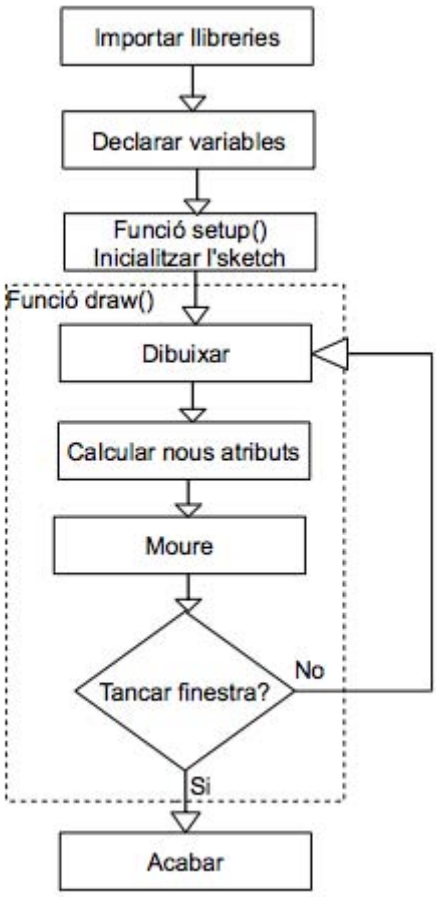


Fig. 1.2 Digrama flux bàsic d'una animació.

Es pot fer una classificació sobre com es vol representar el moviment:

- *Moviment bàsic:* l'objecte va canviant de posició amb el temps. És a dir, a cada frame actualitzes la posició i la matriu de píxels. Això s'aconsegueix cridant la funció background() a cada frame. El que realment estàs fent és pintar el fons novament, per esborrar la posició anterior, i poder pintar després la posició actual.

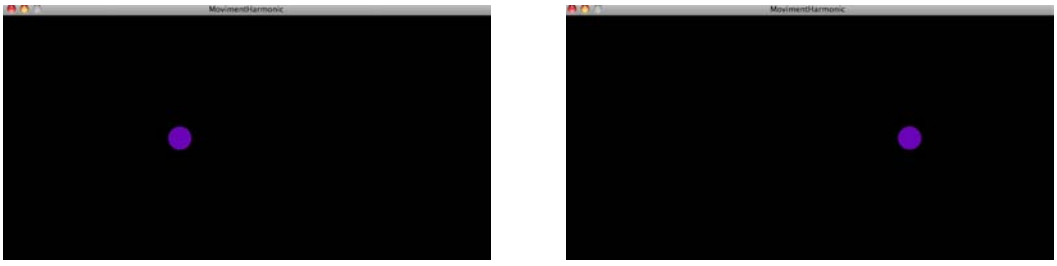


Fig. 1.3 Imatge de moviment bàsic.

- *Moviment i recorregut:* l'objecte va canviant de posició, però deixa el recorregut marcat. La diferència amb l'apartat anterior és que el background() només es crida en el setup(). Per tant, a cada frame es pinta la nova posició, sense esborrar la posició anterior i així s'obté el recorregut que segueix l'objecte.

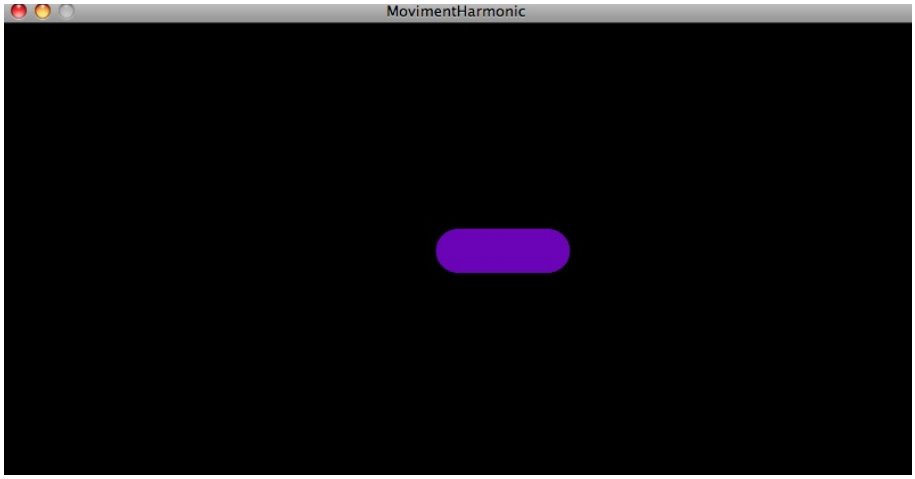


Fig. 1.4 Imatge de moviment i recorregut.

- *Moviment amb estela:* es veu el rastre que deixa un objecte en moviment. Per tal d'aconseguir aquest efecte, no s'ha d'utilitzar la funció background() per canviar el fons, sinó que s'ha de pintar un rectangle de mida igual que la de la finestra i posar-li una certa transparència. A cada frame s'actualitza aquest rectangle i s'obté l'objecte en la nova posició, i l'objecte una mica difuminat en la posició anterior creant una estela. El grau o intensitat de la transparència és un paràmetre que varia de 0 a 255. Com més gran sigui, menys transparència es té i per tant, l'estela serà més petita; en canvi, com més propera a 0 més llarga s'obtindrà l'estela.

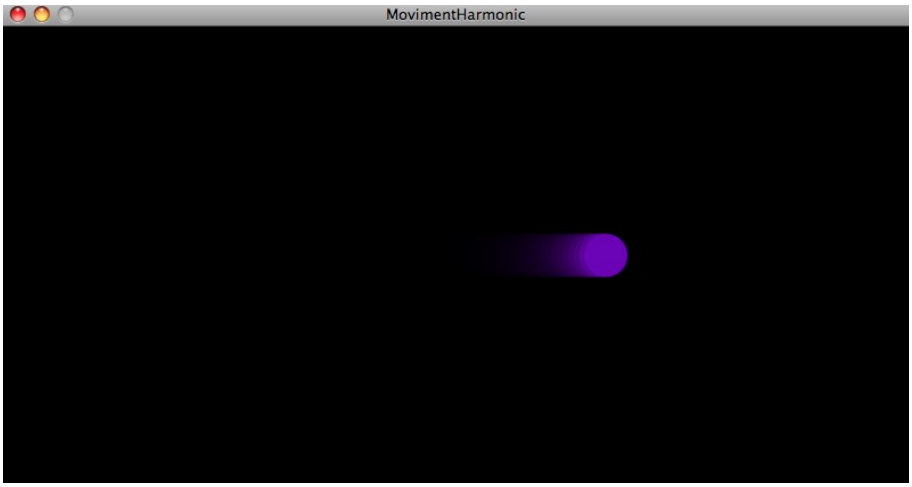


Fig. 1.5 Imatge de moviment amb estela.

Es pot donar el cas que es vol obtenir la trajectòria que seguirà l'objecte sense esperar el temps que tarda en fer tot el recorregut. En aquest cas, s'ha de fer un bucle dins d'un frame que recorri totes les posicions i les pinti.

El moviment que ens interessa més és el real o orgànic. Aquest tipus de moviment també es coneix com a moviment natural, en el sentit de fluidesa, harmonia, llibertat i equilibri. Fa referència a un moviment que no té cap forma precisa, sinó que neix d'un impuls, com a resposta d'una sensació. Es podria dir que en aparença és un moviment anàrquic, però que defineix un estil (caràcter).

Una de les maneres d'aconseguir un moviment elegant és utilitzant la funció sin() ja que ens possibilita generar moviments oscil·latoris d'un objecte. Si es combinen les funcions sin() i cos() es pot aconseguir un moviment més complex, que segueix sent periòdic. A més, per crear certa incertesa o un moviment impredictible es poden combinar les funcions sin() i cos() amb radom().

Aquesta funció random() és la que s'utilitza en Processing per general valors pseudo-aletarois que ens permeten escapar del determinisme computacional.
A la funció se li pot entrar un o dos paràmetres. Si se li entra un paràmetre, el resultat de la funció és un nombre comprès entre 0 i el valor entrat. En canvi, si se li entren dos, el resultat és un valor entre el primer paràmetre i el segon (s'estableix un marge).

En l'apartat següent ens centrem en la programació dels moviments bàsics.

1.2. Tipus de moviment

La cinemàtica és una branca de la mecànica clàssica que estudia els moviments dels cossos sense tenir en compte les seves causes.

Els conceptes bàsics a tenir en compte són els següents:

- *Sistema de referència:* És un conjunt de normes establertes per poder mesurar la posició d'un objecte en el temps i l'espai (magnituds infinites).

  Està format per tres elements:
  - *Punt de referència:* és el punt on es prenen les mesures.
  - *Eixos de coordenades:* el seu origen és el punt de referència. Serveix per determinar la direcció i el sentit del cos en moviment.
  - *Origen temporal:* és l'instant a partir del qual es mesura el temps. Normalment sol coincidir amb l'inici del moviment.

  En Processing la finestra és una matriu de píxels que s'identifiquen per les seves coordenades (x,y). L'origen de coordenades (0,0) o punt de referència, es troba a la part superior esquerra. La coordenada x correspon a l'eix horitzontal mentre que la y a l'eix vertical.

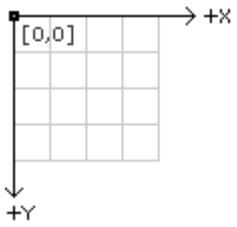


Fig. 1.6 Sistema de referència.

- *Moviment:* és un fenomen físic en el qual els cossos d'un sistema experimenten un canvi de posició en l'espai respecte ells mateixos o amb un altre cos com a referència. Tot cos en moviment descriu una trajectòria.
- *Trajectòria:* és el conjunt de totes les posicions per on passa el cos en moviment.

Utilitzarem les següents magnituds.

- *Temps:* és una magnitud física que mesura la durada o separació d'esdeveniments canviants; és a dir, és el període que transcorre entre l'estat del sistema quan aparenta X i l'instant en què X sofreix una variació per a un observador (punt de referència). En física clàssica, el temps és una variable que cal afegir a l'espai, per tal de poder situar amb precisió qualsevol objecte i la seva història (passat).

- *Espai:* és el conjunt de totes les posicions possibles.

- *Velocitat:* és la variació de la posició en el temps. És una magnitud vectorial, és la mesura vectorial del moviment d'un mòbil, que guarda la direcció del moviment i la rapidesa (mòdul). En cinemàtica podem diferenciar entre velocitat instantània, velocitat mitjana i velocitat angular.

- *Desplaçament:* és un vector que indica la posició del punt o partícula en referència a un origen (posició prèvia). Per tant, el vector va des del punt de referència fins la posició actual. Si un mòbil passa d'una posició inicial x_0 a una posició final x_f, es diu que el mòbil s'ha desplaçat. Es defineix com a vector desplaçament Δx, el vector que uneix x_0 amb x_f.

$$\Delta x = x_f - x_0 \qquad \textbf{(1.1)}$$

- *Acceleració:* és una magnitud física que mesura la variació de velocitat d'un cos respecte el temps. Es tracta també d'una magnitud vectorial. En cinemàtica es troben diferents tipus d'acceleració: acceleració mitjana, acceleració instantània i acceleració tangencial.

S'ha estudiat com programar els diferents tipus de moviments que s'estudien en cinemàtica: moviment rectilini uniforme i uniformement accelerat, tir

parabòlic, moviment circular i moviment harmònic simple; a aquest conjunt de moviments s'ha afegit una petita introducció a la programació de col·lisions. A més, s'han estudiat els models utilitzats per simular un moviment realista, que utilitzen sobretot funcions sinus i cosinus per definir les trajectòries, així com un cert grau d'aleatorietat. Aquests moviments es poden aplicar tant a un objecte individual com a un sistema de partícules.

Per tal de poder observar millor les diferents magnituds que entren en joc en el moviment, cal estudiar primerament moviments bàsics.

1.2.1. Moviment rectilini uniforme (MRU)

Les fórmules que s'utilitzen per aquest tipus de moviment són les següents:

$$x = v_0 t + x_0$$
$$v = v_0 \qquad \textbf{(1.2)}$$

Un moviment és rectilini quan la seva trajectòria és una recta. El moviment serà rectilini uniforme si la velocitat del cos és constant, i per tant, l'acceleració és nul·la. Això implica que la velocitat mitjana entre dos instants qualssevol sempre serà la mateixa. A més, les velocitats instantànies mitjanes seran les mateixes.

Inicialitzem la variable x (posició) amb el valor de x_0 (posició inicial).

$$Per\ t = 0 \rightarrow x = x_0$$
$$Per\ t = 1 \rightarrow x = v + x_0$$
$$Per\ t = 2 \rightarrow x = 2\mathrm{v} + x_0$$
$$Per\ t = 3 \rightarrow x = 3\mathrm{v} + x_0$$

El que s'observa és que cada cop anem sumant la velocitat a la posició anterior.

Per tant, en llenguatge de programació es tindria:

```
int x = x0
int V = v0
void draw()
{
x+=V
...
}
```

1.2.2. Moviment rectilini uniformement accelerat (MRUA)

Aquest moviment es produeix quan un mòbil es desplaça sobre una trajectòria recta estant sotmès a una acceleració constant. És a dir, per a qualsevol instant de temps l'acceleració tindrà el mateix valor.

Com hem vist abans l'espai augmenta amb la velocitat. Tenint una velocitat no constant, sabem que aquesta augmentarà amb l'acceleració. Per tant, tant l'espai com la velocitat dependran del temps.

El que tindrem serà x(t) en funció de x(t-1), v(t-1) i a. Sent x=espai, v =velocitat i a=acceleració.

$$\begin{aligned} x(t) &= x(t-1)+v(t-1) \\ v(t) &= v(t-1)+a \end{aligned}$$

(1.3)

Si es desenvolupen aquestes expressions, tenint en compte el temps discret tenim:

$$\begin{aligned} & x(t)=x(t-1)+v(t-1)=x(t-1)+v(-2)+a= \\ & =x(t-2)+v(t-2)+v(t-2)+a=x(t-2)+2(v(t-2))+a= \\ & =x(t-3)+v(t-3)+2(v(t-3)+a)+a= \\ & =x(t-3)+3\mathrm{v}(t-3)+2\mathrm{a}+a=x(t-3)+3\mathrm{v}(t-3)+(1+2)a= \\ & =x(t-4)+v(t-4)+3(v(t-4)+a)+(1+2)a= \\ & =x(t-4)+4\mathrm{v}(t-4)+(1+2+3)a= \\ & =x(t-5)+v(t-5)+4(v(t-5)+a)+(1+2+3)a= \\ & =x(t-5)+5\mathrm{v}(t-5)+(1+2+3+4)a= \\ & =x(t-6)+v(t-6)+5(v(t-6)+a)+(1+2+3+4)a= \\ & =x(t-6)+6\mathrm{v}(t-6)+(1+2+3+4+5)a=... \end{aligned}$$

(1.4)

Per exemple, si a la darrera expressió substituïm t=6 s'obté:

$$x(t)=x(0)+tv(0)+(1+2+3+4+5)a$$

(1.5)

Observem que l'acceleració està en funció de la suma d'una progressió aritmètica.

La progressió aritmètica és una sèrie de nombres tals que la diferència de dos termes successius qualssevol de la seqüència és una constant. En aquest cas, tenim una progressió aritmètica de constant 1. La suma dels termes d'una

progressió aritmètica s'anomena sèrie aritmètica. La fórmula per la suma dels primers n termes d'una progressió aritmètica és:

$$S_n = \frac{n(a_1 + a_n)}{2} \tag{1.6}$$

Si a les expressions anteriors, tornem t passos endarrere i s'aplica la fórmula de la suma d'una sèrie aritmètica obtenim:

$$x(t) = x(0) + tv(0) + (1+2+3+4+...+(t-2)+(t-1))a$$
$$S_n = \frac{t(a+at)}{2} = \frac{ta}{2} + \frac{t2a}{2} \tag{1.7}$$

Per tant, l'equació de l'espai ens queda:

$$x(t) = x(0) + t(v(0) + \frac{a}{2}) + \frac{t^2 a}{2} \tag{1.8}$$

$$x(t) = x(0) + t\,v(0) + \frac{t^2 a}{2} \tag{1.9}$$

Si ho comparem amb la fórmula estàndard, observem que la nostra velocitat inicial és v(0) +a/2. Aquesta diferència és deguda a la discretització del temps.

Pel que fa a l'equació de la velocitat, fem una taula de valors, per tal d'observar com varia la velocitat a cada unitat de temps.

$$Per\ t = 0 \rightarrow v = v_0$$
$$Per\ t = 1 \rightarrow v = v_0 + a$$
$$Per\ t = 2 \rightarrow v = v_0 + 2\mathrm{a}$$
$$Per\ t = 3 \rightarrow v = v_0 + 3\mathrm{a}$$

En primer lloc, s'observa que la velocitat es comporta igual que la posició en un MRU. És a dir, a cada unitat de temps sumen l'acceleració a la velocitat anterior.

$$v(t) = a_0 t + v_0 \tag{1.10}$$

Per tant, el codi que tindrem serà:

```
int x = x0
int a =  a0;
int v = v0 + a0/2;
void draw()
{
    x += v
    v += a
    ...
}
```

1.2.3. Tir parabòlic (TP)

Es tracta d'un moviment que descriu una trajectòria parabòlica. Pot ser analitzat com la composició de dos moviments rectilinis, un moviment rectilini uniforme en l'eix horitzontal i un moviment rectilini uniformement accelerat en l'eix vertical. L'acceleració del moviment vertical serà la gravetat.

Per tant, el codi que es tindrà serà:

```
int x = x0
int y = y0;
int a =  a0;
int angle = angle0;
int vx = v0cos(angle)
int vy = v0sin(angle) + a0/2;
void draw()
{
    x += vx
    y += vy
    vy += a
    ...
}
```

1.2.4. Moviment circular (MC)

Un moviment circular és aquell moviment la trajectòria del qual és una circumferència. Es basa en tenir un centre de gir i un radi constant. S'introdueixen tres nous conceptes que no han aparegut en moviments anteriors: angle, velocitat angular i acceleració angular. Aquests conceptes són

equivalents a les magnituds del moviment rectilini, però es mesuren en diferents unitats.

Com exemple, si agafem un disc que gira i les seves partícules segueixen una circumferència, és més senzill parlar d'angles recorreguts que de distància. La distància recorreguda per una partícula del disc depèn de la seva posició i és igual al producte de l'angle recorregut pel radi.

Igual que el moviment rectilini, es pot tenir un moviment circular uniforme (en temps iguals recorre el mateix espai) o moviment circular uniformement accelerat.

Per un MCU la velocitat angular serà constant o, el que és el mateix, el mòdul de la velocitat lineal serà el mateix, però la direcció anirà canviant constantment, sent tangent a la trajectòria.

En un MCUA passa el mateix però per a l'acceleració. Es tindrà una acceleració angular constant i la velocitat angular variarà linealment respecte del temps.

En la programació el codi serà:

```
int x = x0
int y = y0;
int v = v0;
int a =  a0;
int angle = angle0;
int eixX = eixX0; //Hem de tenir en compte el punt de gir (eix)
int eixY = eixY0;
void draw()
{
    x = eixX + radi·cos(angle);
    y = eixY + radi·sin(angle);
    v +=a;
    angle += v;
}
```

1.2.5. Moviment harmònic simple (MHS)

Una partícula presenta una moviment harmònic simple si la trajectòria que segueix en funció del temps és una sinusoide.

Per saber la posició de la partícula en cada moment, s'ha d'utilitzar l'equació del moviment. Això és una equació diferencial que explica com evoluciona temporalment el sistema físic corresponent.

En aquest cas, la solució de l'equació diferencial ens dóna l'equació del moviment:

$$y(t) = A\sin(\omega t + \varphi) \tag{1.11}$$

Es pot observar que la posició depèn de diversos paràmetres, com és l'amplitud, la velocitat angular i l'angle inicial.

A més, a partir de l'equació es poden treure unes característiques que tindran tots els MHS.

- El moviment es realitzarà en una regió entre -A i A de l'eix y, ja que els valors màxims i mínims de la funció sinus és -1 i 1.
- Com que la funció sinus és periòdica i es repeteix cada 2π, el moviment també es repetirà cada 2π.

Si volem programar un moviment d'aquest tipus sobre l'eix y, el codi seria:

```
int y = y0;
int v = v0;
int a = a0;
int angle = angle0;
int angleinicial = ai;
void draw()
{
    y = y0+ a·sin(angle+ ai);
    angle += v;
}
```

1.3. Col·lisions

Una col·lisió és una interacció entre dos o més objectes en un petit interval de temps i una regió delimitada de l'espai. En una col·lisió, els dos objectes s'aproximen un a l'altre i interaccionen fortament durant un temps molt curt. És a dir, la col·lisió entre dues partícules es produeix quan una partícula es troba en la seva trajectòria l'altra partícula.

Durant el poc temps que dura la col·lisió, qualsevol força externa és menor que les forces d'interacció entre les partícules, conegudes també com a forces impulsives. Per tant, es menyspreen totes les forces exteriors que actuen sobre el sistema, com pot ser la gravetat. D'aquesta manera, el moment lineal del sistema es manté constant, ja que les forces d'interacció són iguals i oposades. El desplaçament dels objectes durant el temps de col·lisió també es pot menysprear, degut a la brevetat d'aquest fet.

Bàsicament, el que s'observa en una col·lisió és un canvi brusc en el moviment de les partícules que xoquen, o almenys en una d'elles. Per tant, es fàcil poder diferenciar dues situacions: el que passa abans de la col·lisió i el que passa després.

Pel que fa a les col·lisions, se'n poden distingir de dos tipus depenent de la reacció que tenen després del xoc. Si l'energia cinètica total dels dos objectes és la mateixa després de la col·lisió que abans, es tracta d'un xoc elàstic. En canvi, si l'energia cinètica total no és la mateixa després de la col·lisió, es tracta d'un xoc inelàstic. En el segon cas, l'energia perduda es transforma en calor o en una energia interna del sistema. El cas extrem, seria un xoc perfectament inelàstic on després de la col·lisió els dos objectes queden enganxats.

En aquest treball s'ha estudiat els xocs elàstics. Per tant, ens trobem en una col·lisió on no hi ha deformacions dels objectes, ni es perd l'energia en calor. És conserva tant el moment lineal com l'energia cinètica i no hi ha intercanvi de massa entre els cossos, que se separen després del xoc.

Com s'ha dit abans, en el moment de la col·lisió només es tenen en compte les forces d'interacció, per tant, ens trobem en un sistema aïllat. Això significa que en una col·lisió es conserva el moment lineal o quantitat de moviment.

La quantitat de moviment és una magnitud vectorial que es defineix com la massa del cos per la seva velocitat en un instant determinat. Es pot relacionar amb la segona llei de Newton on es diu que si sobre un cos en moviment actua una força, aquesta modificarà el moviment, canviant el mòdul i direcció de la velocitat. La fórmula seria:

$$F = ma = m\frac{dv}{dt} = \frac{dp}{dt} \text{ ; on p = quantitat de moviment}$$

(1.12)

Per a dues partícules que col·lisionen es té:

$$F_{12} = \frac{dp_1}{dt}$$
$$F_{21} = \frac{dp_2}{dt}$$

(1.13)

Per tant, segons la tercera llei de Newton, acció i reacció, per cada força que actua sobre un cos, aquest realitza una força d'igual intensitat i direcció, però de sentit contrari sobre el cos que l'ha produïda. Així, $F_{21} = F_{12}\, i\, p_1 = p_2$ on es demostra que la quantitat de moviment en una col·lisió es conserva.

Per tant, el moment lineal es pot expressar de la següent manera:

$$p = mv$$

(1.14)

La conservació del moment lineal ens proporciona les velocitats finals dels objectes col·lisionats amb les seves velocitats inicials. D'aquesta manera es tindria:

$$m_1 v_{1f} + m_2 v_{2f} = m_1 v_{1i} + m_2 v_{2i} \quad \textbf{(1.15)}$$

Si retornem a la definició de xoc elàstic, trobem que les energies cinètiques inicials i finals són iguals. Si ens fixem en el nostre voltant, les col·lisions no són perfectament elàstiques, sempre hi ha una certa quantitat d'energia que es perd. Per exemple, si una pilota cau sobre una superfície de ciment i rebota fins la seva altura inicial, la col·lisió entre el terra i la pilota és elàstica; però aquest fet encara no s'ha pogut observar. En canvi, a escala microscòpica, les col·lisions elàstiques són més comunes. Per exemple, les col·lisions entre molècules d'aire són sempre elàstiques. En totes les col·lisions elàstiques es té:

$$\frac{1}{2} m_1 v_{1f}^2 + \frac{1}{2} m_2 v_{2f}^2 = \frac{1}{2} m_1 v_{1i}^2 + \frac{1}{2} m_2 v_{2i}^2 \quad \textbf{(1.16)}$$

Es pot fer una simplificació de l'equació, per tal de facilitar la resolució de les velocitats finals. Si s'expressa la velocitat relativa després del xoc en funció de la velocitat relativa abans del xoc s'obté:

$$m_2 (v_{2f}^2 - v_{2i}^2) = m_1 (v_{1f}^2 - v_{1i}^2) \quad \textbf{(1.17)}$$

$$m_2 (v_{2f} - v_{2i})(v_{2f} + v_{2i}) = m_1 (v_{1f} - v_{1i})(v_{1f} + v_{1i}) \quad \textbf{(1.18)}$$

Per tant, en una col·lisió elàstica aplicant la conservació de l'energia i combinant les expressions anteriors s'arriba a:

$$v_{2f} - v_{1f} = -(v_{2i} - v_{1i}) \quad \textbf{(1.19)}$$

A partir de la teoria explicada anteriorment, s'ha pogut calcular les velocitats finals dels objectes a partir de la seva massa i la velocitat abans de la col·lisió.

$$v_{1f} = v_{1i}\left(\frac{m_1 - m_2}{m_1 + m_2}\right) + v_{2i}\left(\frac{2\,m_2}{m_1 + m_2}\right) \quad \textbf{(1.20)}$$

$$v_{2f} = v_{1i}\left(\frac{2\,m_1}{m_1 + m_2}\right) - v_{2i}\left(\frac{m_1 - m_2}{m_1 + m_2}\right) \quad \textbf{(1.21)}$$

En la programació de col·lisions, quan ja s'ha analitzat la física dels objectes i se sap com varien les velocitats, el que ha estat difícil és detectar el moment exacte de la col·lisió sense que ja no s'hagin sobreposat. Això és degut que en la programació el temps és discret, per tant, es pot donar el cas que en un instant de temps t dos objectes no es toquin, però que en l'instant de temps t+1 ja estiguin un damunt l'altre, i no simplement tocant-se. Aquest cas, és la situació més crítica de les col·lisions, i si volem que el moviment sigui net i real hi haurà una feina addicional en la programació.

La conservació del moment lineal i l'energia cinètica també s'aplica en col·lisions en dues dimensions, tenint en compte que la velocitat es descomposarà en una component en l'eix x i una altra en l'eix y. En una dimensió les col·lisions sempre es produiran en la direcció que es mou l'objecte, en canvi, en dues dimensions es pot produir una col·lisió en una direcció diferent a les direccions de moviment dels objectes. Per tant, quan dos objectes que col·lisionen en una direcció diferent a la direcció en la qual s'estan movent, les velocitats s'han de descomposar en una component que segueixi la direcció de la col·lisió i una component normal a la col·lisió. Les equacions per calcular el canvi de de velocitats s'apliquen a les components que segueixen la direcció de la col·lisió, les components normals no canvien.

A continuació, es pot veure una imatge de l'sketch de col·lisions entre dos objectes. En la imatge s'observa el moment de la col·lisió i la velocitat abans del xoc. Les fletxes grogues corresponen a la component normal i la component de la direcció de la col·lisió. La blava representa la direcció cap a on es mou l'objecte.

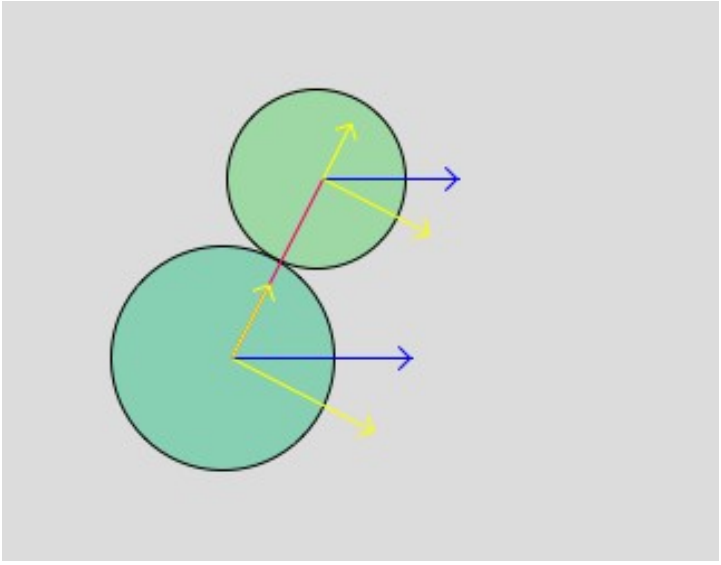


Fig. 1.7 Components velocitat en una col·lisió.

1.4. Exemples implementats

A continuació es troba una llista d'algunes proves que he anat fent aquests mesos. Com que el programa era nou per a mi, he començat amb alguns sketches bàsics on s'observa els tipus de moviments explicats anteriorment. En Processing, fer que l'objecte s'estigui movent és relativament fàcil. Relacionar el moviment programat amb la cinemàtica ha requerit un esforç addicional que, al final, ha donat lloc a un sketch de cada moviment. Tots aquests exemples estan inclosos al CD que s'ha entregat juntament amb aquest document.

- MovimentRectilini.
- MovimentCircular.
- TirParabòlic.
- MovimentHàrmonic.
- Fractal.
  Es tracta d'una peça d'estètica atractiva on es pot observar el moviment circular. Quan es fa click amb el ratolí apareixen més circumferències amb unes petites rodones que segueixen la trajectòria.
- BarraInclinada.
  Es tracta d'una animació on hi ha una barra inclinada enmig de la pantalla i van caient boles. Es veu la fricció amb la barra, l'acceleració que agafen les boles i, evidentment, es detecten les col·lisions entre la barra i les boles, i entre les boles i el marge de la finestra.

- XocSimple.
  Es tracta d'una col·lisió entre dues boles. Quan es pitja la tecla 1 et guarda la imatge, quan les boles xoquen es para i pinta els vectors velocitat, pitjant qualsevol tecla (menys la 1) calcula les noves velocitats i les pinta (però segueix parat) i tornant a pitjar qualsevol tecla (menys la 1) es torna a moure.

- Col·lisions.
  Es tracta d'una animació on hi ha sis pilotes que es van movent per tota la pantalla i es detecten les col·lisions entre elles.

CAPÍTOL 2. SISTEMES DE PARTÍCULES

2.1. Llibreries específiques

Les llibreries són un conjunt de mètodes que amplien les capacitats d'un llenguatge. En el cas de Processing, n'existeixen més d'una cinquantena, la majoria de les quals són contribucions de persones externes del projecte. Una petita part, anomenades “Core Libraries”, són responsabilitat directa dels programadors de Processing i poden, per tant, oferir una mica més de seguretat pel que fa al seu funcionament correcte, però sobretot, pel manteniment i actualització de noves versions.

Per tant, les llibreries de Processing són un recurs on s'ha d'acudir quan es vol estendre les funcionalitats bàsiques del programa. Pel que fa a la seva utilització primerament s'han d'importar al programa i s'han de declarar a l'inici. En un mateix programa es poden utilitzar tantes llibreries com calgui.

En aquest apartat es presenten dues llibreries que s'han estudiat i utilitzat en aquest projecte. Un dels objectius inicials del projecte era l'estudi de la llibreria Physics, que comentem en la secció següent. D'altra banda, també hem usat la llibreria Minim, que permet tractar l'àudio en Processing: importar, reproduir, analitzar i sintetitzar. En aquest cas, només n'hem usat una part petita de les seves funcionalitats, amb una finalitat purament estètica de cara a la peça final. Per això només fem una breu presentació de les funcionalitats utilitzades.

2.1.1. Physics

La Physics és una llibreria creada per Jeffrey Traer Bernstein (http://traer.cc/), que es dedica a relacionar el disseny, la ciència i l'enginyeria per inventar noves formes d'interactuar amb els ordinadors. Actualment, és l'encarregat de “*Human Interface Device Software Prototype group at Apple*”.

La llibreria consisteix en un conjunt de classes per a la simulació de sistemes de partícules. Per tant, ens permet definir sistemes, afegir partícules i crear dos tipus de força entre elles, l'atracció i una força elàstica, com l'exercida per una molla. El que no està implementat són les col·lisions. En el capítol anterior, s'ha parlat de com s'implementarien les col·lisions i la complexitat que comporten.

La llibreria està estructurada en quatre classes. La classe principal és el sistema de partícules (ParticleSystem). Aquesta classe relaciona les altres classes entre sí, i és la que fa que la simulació avanci. Les altres són Particle, Attraction i Spring. Aquesta llibreria dóna l'opció de crear el sistema en 2D o 3D. En aquest treball, sempre s'ha treballat en dues dimensions (2D).

Per tant, al començament del programa, si es vol utilitzar aquesta llibreria, s'ha de definir i configurar el sistema de partícules. Quan es crea el sistema, se li han d'entrar dos paràmetres, la gravetat i el fregament. Si es vol un sistema que simuli la realitat, la gravetat estarà a l'eix y, però Processing dóna l'opció de crear gravetat en els tres eixos (x, y, i z). Això ens permet obtenir un sistema totalment propi. Un altre avantatge que presenta és que es poden crear tants sistemes com es vulgui, cadascun amb una configuració diferent.

Un sistema permet crear diferents partícules i forces entre elles. Per tal de manejar-ho, es té un objecte per a cada partícula o força creada. A més, a mesura que avança la simulació es poden canviar els paràmetres del sistema. És a dir, pots augmentar o disminuir la gravetat quan et convingui, es pot eliminar el fregament en un moment donat o fins i tot canviar la velocitat amb què avança la simulació.

Pel que fa la classe Particle, les partícules representen objectes amb formes no definides. És a dir, es crea un objecte amb els seus paràmetres però no es dibuixa. Això té l'avantatge que es poden tenir partícules invisibles i que si es volen pintar se'ls pot donar la forma i la mida que es vol (rodona, quadrada, triangular, etc). Cada partícula es caracteritza per quatre propietats: la massa, l'edat, la posició, i la velocitat que té associada.

Com que cada partícula pertany a un sistema, es pot triar si la partícula es veurà afectada per les forces del sistema o no. Per això es defineixen els paràmetres fixa (la partícula estarà quieta i no es veurà afecta per cap força externa) o lliure. I per últim, també es pot jugar amb el temps de vida de la partícula, que en qualsevol moment es pot destruir. A més, se li pot especificar un tipus de moviment a cada partícula i durant la simulació, anant canviant de velocitats.

Com s'ha explicat anteriorment, les altres dues classes representen els dos tipus de força que hi ha implementats en aquesta llibreria. S'ha de tenir en compte que només permet crear forces entre parells de partícules.

La classe Spring serveix per intentar mantenir la mateixa distància entre dues partícules. Per tal d'aconseguir-ho es crea una força que les empenyerà per tal d'aproximar-les o allunyar-les.

Per crear aquest fenomen se li entren cinc paràmetres: la primera partícula, la segona partícula, la força, el coeficient d'amortiment i la distància. El paràmetre de distància és la distància que volem que les partícules mantinguin entre elles. Com més gran és la força, més poc tardarà en aconseguir la distància desitjada. A més, com més gran sigui l'amortiment, menors seran les oscil·lacions que definiran la seva trajectòria. És a dir, si el coeficient d'amortiment és petit, les dues partícules aniran oscil·lant fins a trobar-se a la distància desitjada.

Per acabar, la classe Attraction, fa referència a les forces d'atracció (força positiva) o repulsió (força negativa). La magnitud de la força dependrà de la distància on es troben les dues partícules que s'atreuen o repel·len i de les

masses de les partícules, tal com estableix la llei de gravitació universal. La força ve definida com:

$$F = \frac{G\, m_1 m_2}{d^2} \qquad \textbf{(2.1)}$$

Quan es crea una força d'aquest tipus, se li entren quatre paràmetres: la primera partícula, la segona partícula, la força, i la distància mínima. Aquesta distància mínima és la distància a partir de la qual la força d'atracció ja no augmentarà més.

Igual que a les altres classes, mentre la força d'atracció actua, en la simulació es poden variar alguns paràmetres com la magnitud de la força i la distància mínima. A més, hi ha l'opció d'activar-la i desactivar-la. Aquest fet de poder canviar els paràmetres mentre la simulació està en marxa, sempre ajuda a tenir més joc a l'hora de crear l'sketch.

A continuació, s'explicaren en detall els mètodes i atributs que s'han utilitzat per fer proves amb la llibreria i l'sketch final. En la figura següent hi ha representat el diagrama de classes d'aquesta llibreria.

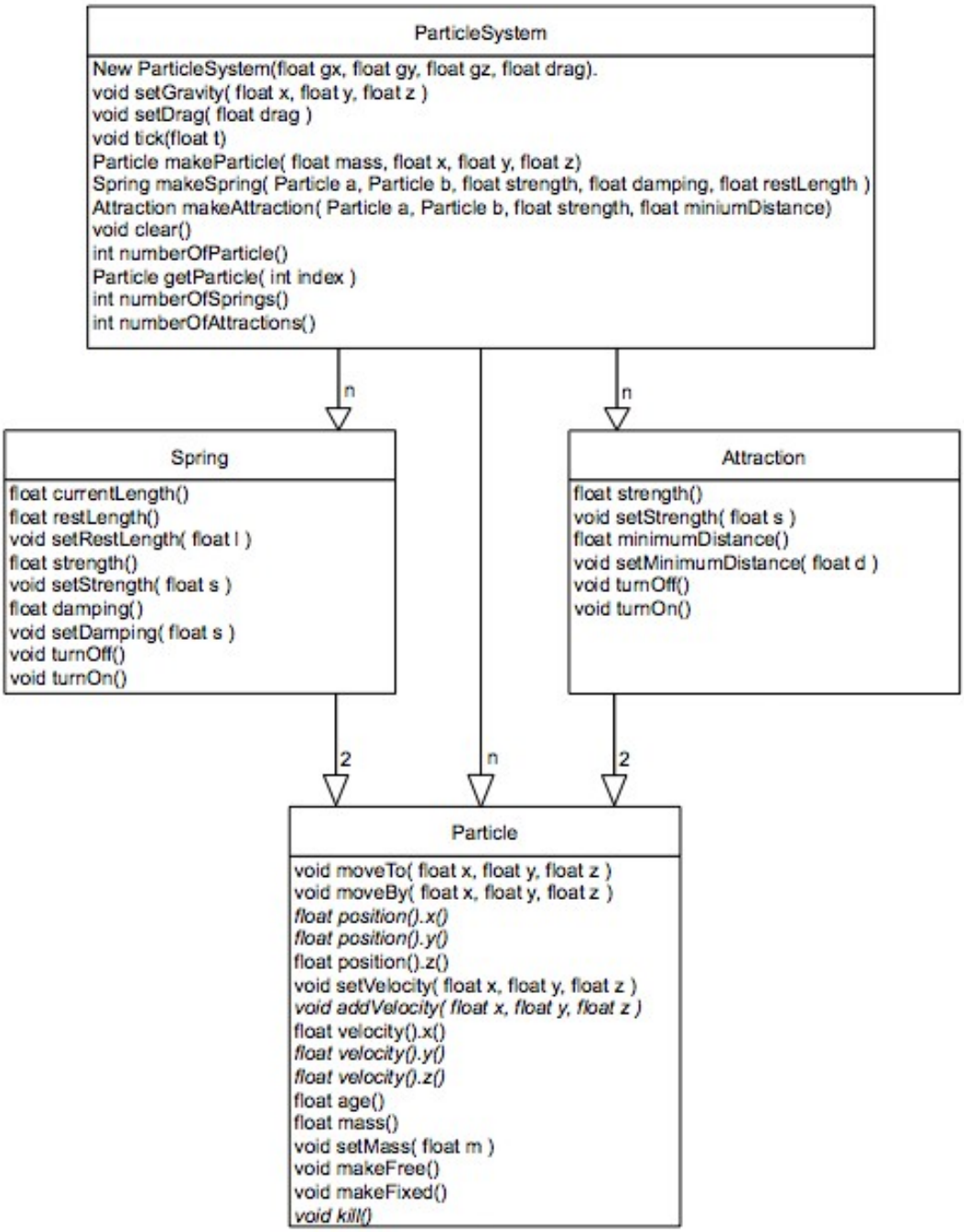


Fig. 2.1 Diagrama de classes.

2.1.1.1. ParticleSystem

Les funcions públiques d'aquesta classe són: la creadora del sistema, un conjunt de funcions que modifiquen els atributs del sistema, incloent funcions per afegir partícules, atracció i forces elàstiques, i les funcions de consulta d'atributs.

new ParticleSystem(float gx, float gy, float gz, float drag).

Aquesta funció serveix per crear el sistema de partícules. Se li entra la gravetat en les tres direccions i el fregament. També hi ha l'opció d'entrar-li dos paràmetres que corresponen a la gravetat en l'eix y i el fregament.

void setGravity(float x, float y, float z).
Aquesta funció serveix per donar gravetat al sistema. També se li pot entrar un sol paràmetre i que correspondria a la gravetat de l'eix y.

void setDrag(float drag).
Aquesta funció serveix per afegir fregament al sistema.

void tick(float t).
Aquesta funció és molt important ja que fa que la simulació avanci en un temps t. S'ha de cridar dins el draw() i, per defecte, t=1.0. Si es vol una acceleració o desacceleració dels objectes s'hauria d'anar canviant el paràmetre t.

Particle makeParticle(float mass, float x, float y, float z).
Aquesta funció serveix per crear partícules dins el sistema. Per defecte la massa val 1 i es crea a la coordenada (0,0,0).

Spring makeSpring(Particle a, Particle b, float strength, float damping, float restLength).
Aquesta funció serveix per crear un Spring, és a dir, una força elàstica, entre dues partícules que ja han estat creades.

Attraction makeAttraction(Particle a, Particle b, float strength, float miniumDistance).
Aquesta funció serveix per crear una força d'atracció o repulsió entre dues partícules. Si la força que se li entra és positiva crearà atracció, si es negativa crearà una força de repulsió.

void clear().
Aquesta funció serveix per eliminar totes les partícules i forces creades en el sistema. Es com si es tornés a la configuració inicial, amb una certa gravetat i fregament.

int numberOfParticle().
Aquesta funció retorna el nombre de partícules que té un sistema.

Particle getParticle(int index).
Aquesta funció retorna la partícula amb l'index especificat.

int numberOfSprings().
Aquesta funció retorna el nombre de forces elàstiques que té un sistema.

int numberOfAttractions().
Aquesta funció retorna el nombre d'atraccions que té un sistema.

2.1.1.2. Particle

La partícula és l'element bàsic del sistema. Les funcions públiques d'aquesta classe són: un conjunt de funcions que modifiquen els atributs de la partícula, incloent posició, velocitat, massa, edat, atribut de fixa o no fixa, i les funcions de consulta d'atributs. No hi ha una funció creadora pública, perquè les partícules es creen des del sistema.

void moveTo(float x, float y, float z).
Aquesta funció serveix per moure la partícula a la posició (x,y,z).

void moveBy(float x, float y, float z).
Aquesta funció serveix per moure la partícules a partir d'increments en la posició actual. És equivalent a tenir una velocitat a cada eix.

float position().x().
float position().y().
float position().z().
Aquestes tres funcions serveixen per obtenir les posicions d'una partícula.

void setVelocity(float x, float y, float z).
Aquesta funció serveix per donar velocitat a la partícula.

void addVelocity(float x, float y, float z).
Aquesta funció serveix per afegir velocitat a la partícula. Els paràmetres que se li entren se sumaran a la velocitat que ja tenia la partícula. D'aquesta manera s'aconsegueix acceleració i desacceleració.

float velocity().x().
float velocity().y().
float velocity().z().
Aquestes tres funcions retornen la velocitat d'una partícula, en cadascun dels eixos.

float age().
Aquesta funció retorna l'edat de la partícula. L'edat s'incrementa cada vegada que avança la simulació.

float mass().
Aquesta funció retorna la massa de la partícula.

void setMass(float m).
Aquesta funció serveix per modificar la massa de la partícula. Quan s'apliquen forces s'ha de tenir en compte que com més massa tingui la partícula més gran haurà de ser la força per tal de moure-la.

void makeFree().
Aquesta funció serveix per fer lliure la partícula. Això significa que es veurà afectada per totes les forces del sistema.

void makeFixed().
Aquesta funció serveix per fer fixa la partícula. Per tant, la partícula es quedarà en un lloc i no es veurà afectada per cap força.

void kill().
Aquesta funció serveix per destruir la partícula i totes les forces que tenia associades a ella.

2.1.1.3. Spring

Les funcions públiques d'aquesta classe són: un conjunt de funcions que modifiquen els atributs de la força elàstica, incloent distància, força, coeficient d'amortiment, atribut d'activació o desactivació, i les funcions de consulta d'atributs.

float currentLength().
Aquesta funció retorna la distància actual.

float restLength().
Aquesta funció retorna la distància que es vol que mantinguin les dues partícules.

void setRestLength(float l).
Aquesta funció serveix per definir la distància entre les dues partícules.

float strength().
Aquesta funció retorna el valor de la força elàstica.

void setStrength(float s).
Aquesta funció serveix per definir el valor de la força elàstica.

float damping().
Aquesta funció retorna el coeficient d'amortiment.

void setDamping(float s).
Aquesta funció serveix per donar valor al coeficient d'amortiment.

void turnOff().
void turnOn().
Aquestes dues funcions serveixen per activar o desactivar l'efecte de la força elàstica.

2.1.1.4. Attraction

Les funcions públiques d'aquesta classe són: un conjunt de funcions que modifiquen els atributs de la força d'atracció, és a dir, distància, força i atribut d'activació o desactivació, i les funcions de consulta d'atributs.

float strength().
Aquesta funció retorna el valor de la força que s'utilitza per crear atracció o repulsió.

void setStrength(float s).
Aquesta funció dóna valor a la força. S'ha de tenir en compte que per crear atracció la força ha de ser positiva i, per repulsió ha de ser negativa.

float minimumDistance().
Aquesta funció retorna la distància a partir de la qual la força d'atracció ja no augmenta.

void setMinimumDistance(float d).
Aquesta funció serveix per donar valor a la distància mínima.
void turnOff().
void turnOn().
Aquestes dues funcions serveixen per activar i desactivar l'atracció.

2.1.1.5. Exemples implementats

Quan s'ha anat agafant pràctica amb el programa, s'ha començat a treballar amb les llibreries, sobretot la que ens interessava més, la Physics. En aquest cas, per aconseguir algun sketch decent, s'han hagut de fer moltes proves per poder saber quins valors dels paràmetres són els adequats per tal de generar unes forces reals.

- AgafaPlujaAtraccio.
  Pitjant i arrossegant el ratolí es va movent la pilota vermella. Les partícules negres tenen atracció cap a la pilota que va augmentant de mida cada vegada que agafa una partícula.

- Atrac_Rep.
  Les partícules que apareixen tenen una certa repulsió entre elles i atracció cap el centre de la finestra. Si es pitja i s'arrossega el ratolí apareix una pilota vermella que repel·leix amb gran intensitat les altres partícules.

- Planeta.
  És una prova amb la llibreria Physics on utilitzo la classe Spring. Tot i que Spring està relacionada amb l'amortiment, jo l'he utilitzat com una força que intenta mantenir la mateixa distància entre les dues partícules.

2.1.2. Minim

Com que s'ha incorporat àudio a la peça final per fer-la més atractiva, hem hagut d'incorporar una llibreria que ens permetés fer sonar un fitxer d'àudio i fer interactuar els gràfics amb el senyal.

De totes les llibreries de Processing que permeten incorporar àudio, s'ha escollit la Minim per la seva facilitat d'ús i perquè està inclosa en el "core" del programa. El seu objectiu és que la integració d'àudio en l'sketch es faci de la manera més simple possible, però proporcionant, a la vegada, una certa flexibilitat als usuaris més avançats. La llibreria Minim està formada per 64 classes diferents dividides en: 34 classes dedicades a funcions generals, 4 classes dedicades a l'anàlisi d'àudio, 8 classes dedicades a efectes i 8 classes dedicades a diferents tipus de senyals.

AudioPlayer player.
La variable player és on es carrega la cançó.

BeatDetect beat.
La variable beat serveix per detectar tots els beats de l'àudio.

void loadFile(string fileName, int size).
Aquesta funció carrega l'arxiu d'àudio que es vol posar a l'sketch i defineix la mida del buffer.

void setSensitivity(int n).
Aquesta funció s'utilitza per amortir l'anàlisi. Si se li entra una sensibilitat de 300, cada vegada que l'algoritme detecti un beat, s'haurà d'esperar 300 milisegons abans de buscar el següent beat.

void play().
Aquesta funció reprodueix l'arxiu d'àudio.

void close().
Aquesta funció tanca la cançó.

2.2. ProcessingBCN

ProcessingBCN ha estat la primera trobada de Processing a Barcelona. L'organització d'aquest esdeveniment va ser a càrrec de Lali Barrière i Alba G. Corral. La Lali em va proposar que ajudés en l'organització i en la possibilitat de presentar el que estava fent en el TFC. A més, vaig tenir la sort que l'escola em va donar una beca per col·labora-hi.

La trobada estava oberta a qualsevol persona que treballés o tingués algun interès en Processing ja siguin artistes, dissenyadors, programadors, etc. La idea principal era fer una trobada presencial i compartir experiència i

coneixements o simplement, explicar el que estàs fent i veure el que fan els altres.

Aquesta trobada va ser el 23 i 24 de Maig a Hangar. L'Hangar és un centre per a la producció i la recerca artística fundat el 1997 per l'Associació d'Artistes Visuals de Catalunya (AAVC) per donar suport a creadors i artistes i oferir serveis que s'adaptin a les necessitats de producció que sorgeixen en el món de la creació. L'espai està subvencionat principalment per la Generalitat de Catalunya i l'Ajuntament de Barcelona. El seu model de gestió i la seva filosofia, enfocada a la producció i a la investigació, fa d'Hangar un espai únic.

El centre té la seu a un edifici industrial rehabilitat al barri del Poblenou de Barcelona, que dóna cabuda a 15 tallers individuals, un medialab, 2 platós, un servei de lloguer d'equipament, tècnics i assessorament de producció. A més a més, Hangar organitza un programa de tallers de formació per a artistes, un programa d'intercanvis internacionals i ofereix beques de producció. Aquest espai va ser cedit a l'organització de ProcessingBCN.

Pel que fa a l'estructura de la trobada, es va creure que el més indicat era algunes xerrades sobre temes concrets relacionats amb Processing i que puguin ser d'interès per a la majoria, complementades per sessions més o menys lliures, dedicades a la programació en grup o intercanvi de projectes entre els participants.

A continuació es troba un esquema on es detalla l'estructura de la trobada. La xerrada del segon dia, sobre moviment, és la presentació que vaig fer jo. A l'annex B es trobarà la presentació utilitzada per aquesta ocasió.

Dissabte 23
11h - 13h > Sessió matinal: Interacció – multitouch
13h - 14h > Dinar
14h - 20h > Code Kitchen - Laboratori de programació

Diumenge 24
11h - 13h > Sessió matinal: Moviment
13h - 14h > Dinar
14h - 18h > Code Kitchen - Laboratori de programació
18h - 20h > Presentacions - actuacions - fi de la trobada

Pel que fa al primer dia, al matí tocava la xerrada sobre multitouch a càrrec Oriol Aragonés (director tècnic de Ping Pong Technologies, S.L.). La xerrada va ser molt interessant i ben plantejada. Ens va ensenyar alguns projectes per tal de veure Processing com una eina poderosa que es pot utilitzar per publicitat, per presentar algun esdeveniment, fer campanyes, etc. Una cosa que va agradar molt va ser com utilitzar el moviment agafat des de una càmera de vídeo o una webcam en Processing. A més, va ensenyar un simulador per si algú no disposa de cap càmera de vídeo i una manera de fer una taula multitouch casolana i senzilla.

A la tarda, cadascú va començar a programar i comentar tots els dubtes que tenia, errors que havia trobat i que no sabia perquè passava, etc. L'Oriol va ser de gran ajuda ja que coneixia molt bé la majoria de llibreries utilitzades per a multitouch i altres temes d'interacció.

El diumenge al matí va tenir lloc la xerrada sobre moviment. Sabent que a la xerrada hi havia gent que es dedicava a l'art, l'objectiu de la presentació va ser relacionar els fonaments matemàtics i físics de moviment amb la programació.
L'estructura de la presentació va ser molt semblant a aquest treball. Al principi, es va explicar una mica el funcionament del programa i les coordenades cartesianes i polars. Es suposà que la gent ja havia programat moviment en Processing, però la intenció de la xerrada era que quedés clar el que realment es fa. Per tant, es va partir de la idea en que a cada frame el que es fa és calcular la posició, dibuixar l'objecte i moure, i les diferents opcions que hi ha, ja sigui dibuixar la trajectòria que seguirà l'objecte, deixar una estela o tenir un moviment net.

A continuació es van explicar els diferents tipus de moviment com rectilini, circular i tir parabòlic relacionat-ho amb la cinemàtica. També es va tocar el tema de l'ús de les matemàtiques per obtenir corbes. A més, es va donar una petita pinzellada sobre com programar col·lisions, on em vaig adonar de que diverses persones ho havien intentat, i que sempre et trobes amb algun petit problema que fa que l'sketch no acabi de funcionar.

Per acabar, vaig explicar el funcionament de la llibreria Physics. Per explicar-ho, es va obrir directament la pàgina web de la llibreria ja que està ben estructurada i proporciona la llista de totes les funcions que estan implementades. Es van mencionar les coses importants per tal de començar a utilitzar-la com per a què serveix, el que et permet i no la llibreria, els problemes que m'he trobat, etc.

Juntament amb la presentació, es va lliurar una sèrie d'exemples on es veien implementades les coses que s'havien explicat.

Per la tarda es va tornar a fer el code kitchen on hi va haver gent que va seguir amb el multitouch i altres que van començar a fer proves de programació de moviment i amb la llibreria. Per finalitzar la trobada, es va fer un concert, acompanyat de visuals en Processing.

A continuació hi ha dues fotografies de ProcessingBCN. La primera representa quan es va muntar una taula tàctil casolana i posant el dit a sobre i movent-lo a la pantalla de l'ordinador ja es veu que s'ha detectat. L'altre foto, hi sortim el Sergi (company de la universitat que fa el seu TFC sobre Processing) i jo.

Fig. 2.2 Fotografia taula tàctil.

Fig. 2.3 Fotografia Sergi i jo a ProcessingBCN.

2.3. Programació d'un sketch

Un dels objectius d'aquest treball ha estat crear una peça final on es veiessin reeflectits els conceptes i tècniques apresos durant aquests mesos. Aquest objectiu inclou un primer pas de decidir què fer, ja que es tracta d'una feina

creativa. Al principi no sabia ben bé què fer: l'objectiu era, bàsicament, "aconseguir una peça que ens agradi".

La meva tutora em va explicar que estava implicada en un projecte que es deia ARU, i a partir d'aquí, va sortir la idea d'un sketch on hi hagués partícules en moviment que anessin formant la paraula ARU.

En la implementació de la peça final podem distingir els elements següents:

- Idea o argument de l'animació: la formació de la paraula ARU.
- Programació dels gràfics: la feina més important, on s'han aplicat els coneixements adquirits prèviament.
- Aspectes estètics: algorismes d'animació molt similars poden donar lloc a peces molt diferents, només amb petits canvis com poden ser les formes dels objectes dibuixats, els colors... En aquest sentit hem vist que s'han de fer bastantes proves per tal d'aconseguir efectes que ens satisfacin estèticament.
- Interacció amb música: amb l'objectiu de fer la peça estèticament més interessant. Tot i que s'ha programat una interacció bastant senzilla, ha representat una feina addicional, perquè s'ha hagut d'estudiar una nova llibreria que no estava prevista en els objectius inicials del projecte.

Així, la peça final es basa en la creació a poc a poc de partícules, agrupades en diferents grups, que es van movent per la pantalla. Fins que no totes les partícules d'un grup han nascut, aquest grup es manté quiet. A més, fins que no totes les partícules estan creades no comença la interacció amb la música. Finalment, les partícules acaben formant la paraula ARU.

Un cop la idea de la peça final va estar definida, es van començar a fer proves amb el nombre de partícules, els moviments que portarien, el color, etc. El color té una gran importància sobre el resultat final, ja que l'elecció d'una bona combinació de colors pot fer molt atractiu l'sketch.

La música també és un factor que ajuda a que la peça final sigui atractiva. Per aquest motiu, i gràcies a que la llibreria Minim és bastant simple, s'ha pogut posar música a l'sketch i interaccionar amb ella. Per aconseguir-ho s'ha utilitzat la classe BeatDetect que ens permet detectar beats o cops en bandes de freqüència diferents: són el Kick, Snare i Hat, que prenen el nom d'elements diferents d'una bateria. L'efecte obtingut és que cada vegada que es detecta un d'aquests beats, dues partícules aleatòries augmenten la seva mida significativament.

Pel que fa el moviment, la primera idea que es tenia era que totes les partícules es moguessin cap al mateix lloc, és a dir, que tinguessin el mateix punt final de la seva trajectòria. Per tant, s'han calculat unes equacions per definir aquest moviment, però a la vegada amb una certa aleatorietat. Al final, s'han definit fins a quatre moviments diferents amb transicions d'un a l'altre. Quan s'arriba al darrer, torna a començar el primer moviment calculat.

A més, per no notar el canvi d'un moviment a l'altre, s'ha decidit que no totes les partícules canviarien de moviment. És a dir, quan s'ha de canviar el

moviment, si es tenen 200 partícules, s'agafaran 150 partícules aleatòries que passaran al moviment següent, mentre que les altres quedaran en el mateix.

Per últim, quan les partícules porten prou temps de vida, s'activa l'atracció. Per a l'atracció, s'han creat unes partícules fixes que formen la paraula ARU i que no es pinten. Així, l'atracció s'ha fet entre una partícula invisible de la paraula ARU i una partícula que es va movent. D'aquesta manera, les partícules s'aniran movent cap al centre de la pantalla per formar la paraula ARU. A més, degut a les força d'atracció, les partícules es ralentitzen una mica. Això ens ha obligat a augmentar les velocitats de les partícules que no es veuen atretes, multiplicant-les per 4.

Quan alguna partícula s'ha quedat enganxada i es començar a formar la paraula, s'ha creat una espècie de fum que surt de la paraula. Aquestes partícules de fum, estan en un sistema diferent on la gravetat s'ha posat negativa, per això van cap amunt. A cada frame es crea més fum, se li posa velocitat i es mira l'edat de la partícula. Com més vella sigui, es pintarà de color més fluix (serà més transparent) fins que arribi a una edat de 40 que es destruirà la partícula.

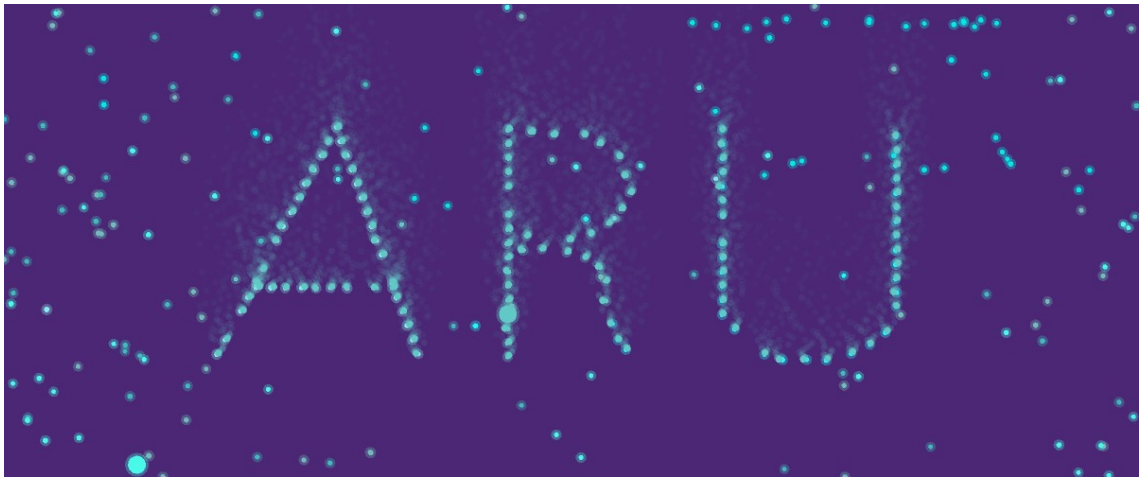
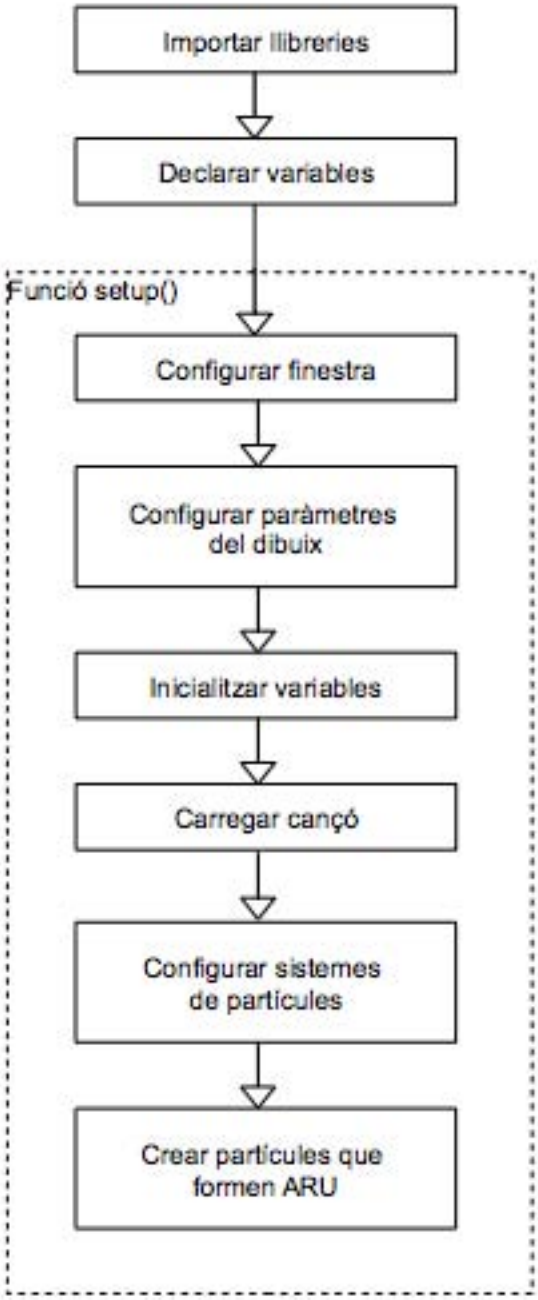
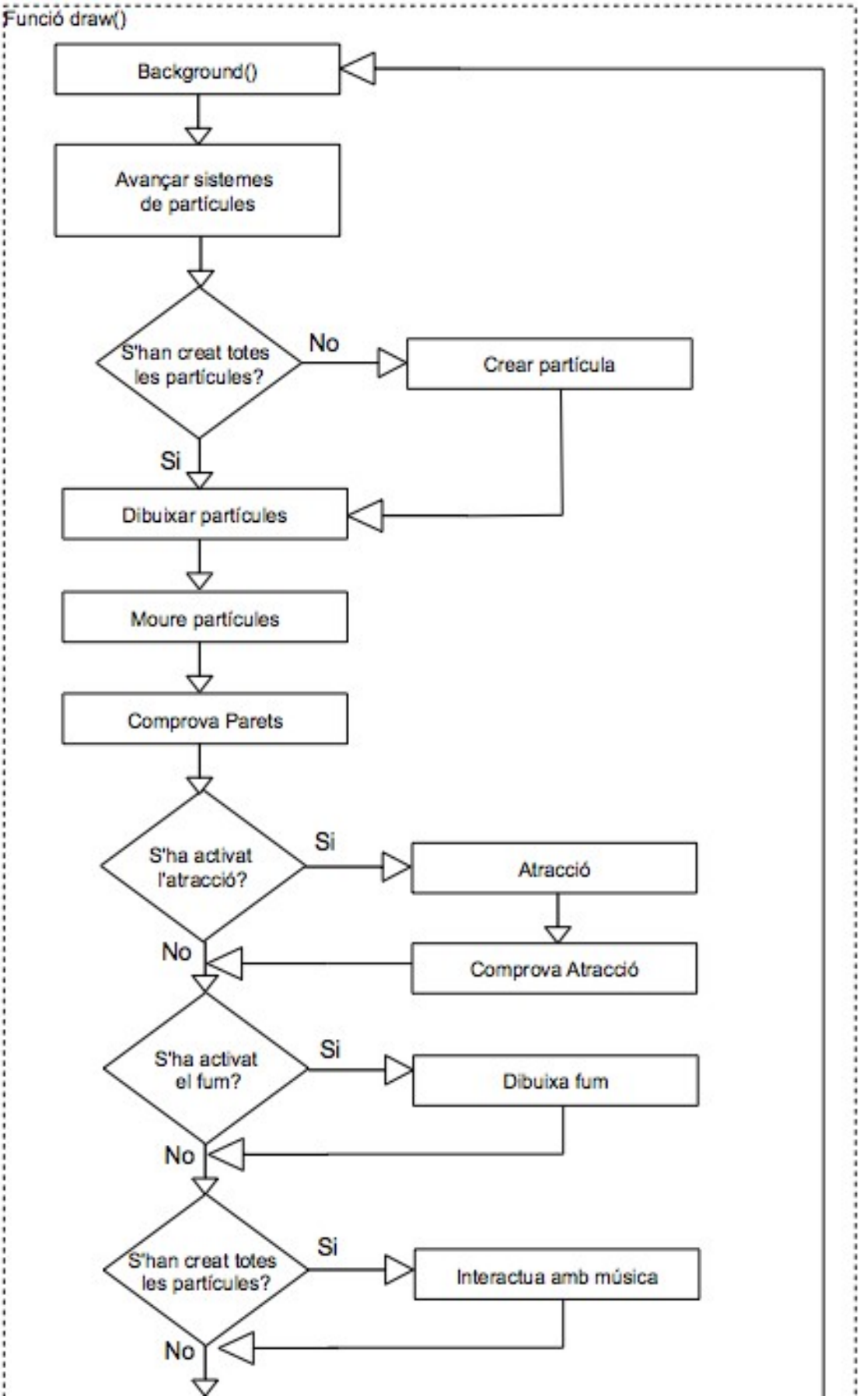


Fig. 2.4 Imatge de l'sketch.

L'sketch té una estructura de quatre classes. L'ARU és la classe on estan guardades les partícules fixes que formen la paraula. La classe BeatListener et permet escoltar els tons de la música i detectar els que ens interessen. L'altre classe és el Moviment, on estan guardades totes les equacions dels diferents moviments. I finalment, es té la classe principal on hi ha definides totes les variables, es tenen els mètodes setup() i draw() habituals de totes les animacions en Processing, i les altres funcions que s'utilitzen per obtenir la peça final. A continuació, hi ha el diagrama de flux de la classe principal.


Importar llibreries
Declarar variables
Funció setup()
Configurar finestra
Configurar paràmetres del dibuix
Inicialitzar variables
Carregar cançó
Configurar sistemes de partícules
Crear partícules que formen ARU


Funció draw()
Background()
Avançar sistemes de partícules
S'han creat totes les partícules?
No
Crear partícula
Si
Dibuixar partícules
Moure partícules
Comprova Parets
S'ha activat l'atracció?
Si
Atracció
No
Comprova Atracció
S'ha activat el fum?
Si
Dibuixa fum
No
S'han creat totes les partícules?
Si
Interactua amb música
No

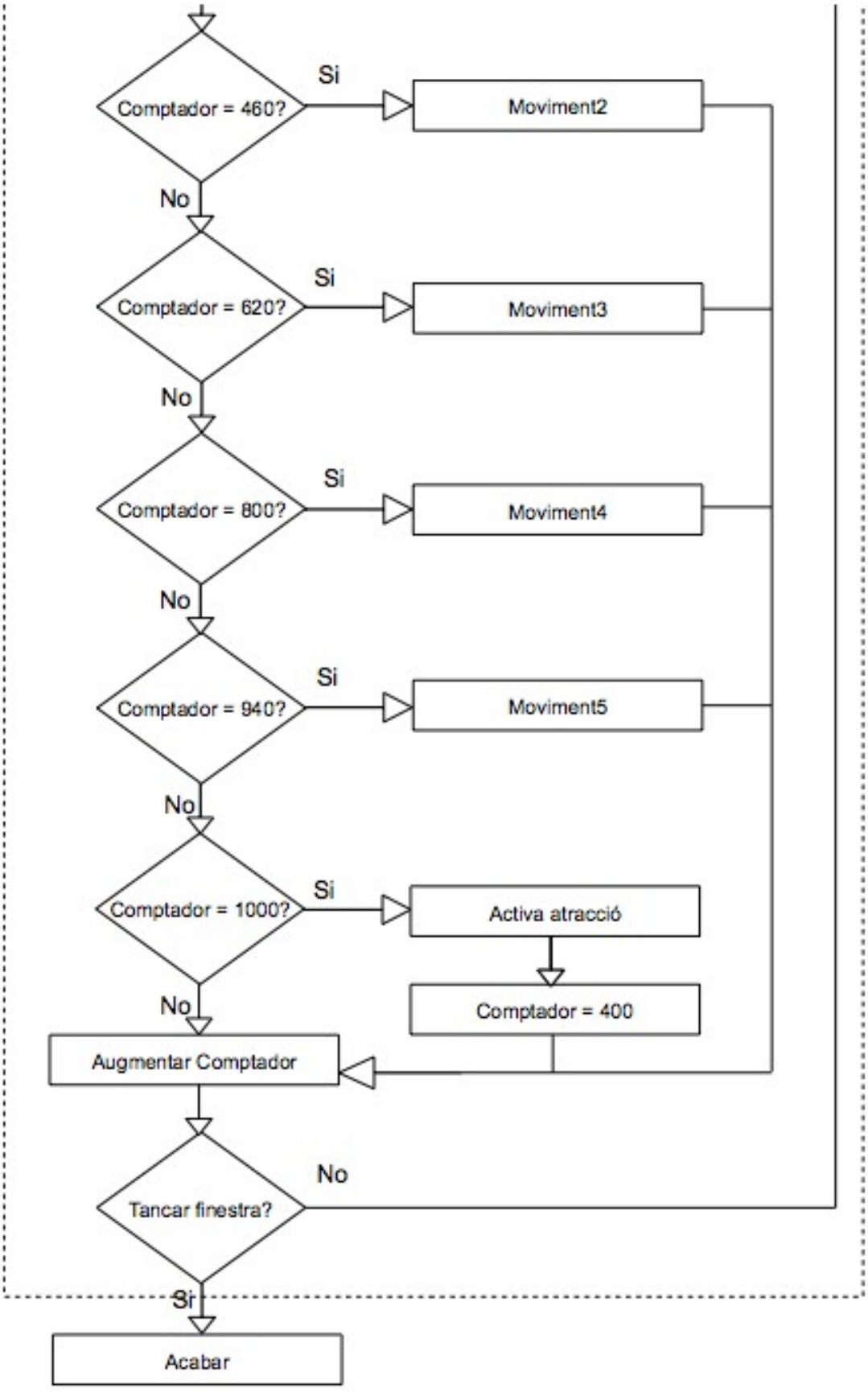


Fig. 2.5 Diagrama de flux classe principal.

CONCLUSIONS

Aquest és un projecte de programació gràfica en el qual s'ha estudiat la programació de moviment i la simulació de sistemes de partícules, tenint en compte les lleis de la física. S'emmarca en el context de l'ús creatiu de la programació, és a dir, l'aplicació de la tecnologia a l'art. Això fa que no es tracta d'un projecte d'enginyeria pura, amb la dificultat afegida del treball creatiu. Però també amb l'interès de treballar amb metodologies diferents i de veure noves aplicacions de l'enginyeria.

S'ha utilitzat el programa Processing que és un projecte col·laboratiu de codi obert. Això, ha permès aprendre un nou entorn de programació i un nou llenguatge, el Java. Per tant, durant les primeres setmanes l'objectiu del treball ha estat aprendre a manejar el programa, seguint tutorials i consultant diversos llibres.

El pas següent ha estat centrar-se en els objectius del treball. Estudiar i analitzar la programació de moviment real i orgànic. Primerament, s'han analitzat els moviment més bàsics relacionant-ho amb les lleis de la cinemàtica. Després s'ha treballat per aconseguir moviments més harmònics. Per obtenir un moviment més harmònic, s'havia suposat que la funció random() aniria bé per afegir una mica d'aleatorietat dintre d'un moviment definit. Però, el que realment s'obté són uns canvis bruscs en la trajectòria. No s'ha aconseguit l'harmonia que s'esperava.

Juntament amb el moviment, es va començar a estudiar les col·lisions. Pel que fa a la detecció si l'objecte xoca en la paret (marge de la finestra) o amb un altre objecte quiet no suposa gran dificultat. Només s'ha de triar què és vol fer quan passa. En canvi, les col·lisions entre dos o més objectes han representat una part difícil d'aquest treball per la seva complexitat a l'hora de programar-les. Tot i això, em sembla que el treball realitzat amb les col·lisions és bastant satisfactori.

Sobre el tema de les col·lisions m'ha cridat l'atenció que no estiguessin implementades en cap llibreria, alhora que molta gent hi està interessada i ha intentat programar-ho. La majoria de documentació que es troba tracta col·lisions de dos objectes i de manera limitada.

La simulació de sistemes de partícules s'ha estudiat mitjançant la llibreria Physics. La primera impressió que em vaig endur sobre aquesta llibreria va ser molt positiva. L'autor dóna una sèrie de consells útils i explica cada funció. Tot i això, quan vaig començar a fer proves em vaig donar compte que no era tant fàcil com semblava, ja que s'han d'ajustar molts de paràmetres. Per exemple, per crear la força has de tenir en compte les masses de les partícules, la magnitud de la força, la distància entre elles, la configuració del sistema, etc. El control d'aquests paràmetres requereix moltes proves. Quan ja comences agafar pràctica, t'adones del bon resultat que pots obtenir i que realment totes les funcions són bastant útils.

Finalment, el darrer dels objectius inicials del projecte era crear una animació on s'utilitzessin les eines adquirides. S'ha de dir que tot i ser un objectiu interessant, la programació d'un sketch requereix la cerca d'una idea inicial, que a vegades pot costar. A més, depèn del gust de cada persona. És a dir, quan ja es sap el que es vol programar i obtenir, s'ha d'anar jugant amb els colors, les formes i altres aspectes visuals que només s'aconsegueixen a base de proves. Una idea que s'ha utilitzat aquí és posar música a la peça final per tal de fer-la més atractiva, cosa que ha representat una feina addicional.

Per a concloure, les eines apreses en la primera etapa del projecte han estat presentades a la trobada ProcessingBCN. M'agradaria remarcar que l'experiència de participar a ProcessingBCN ha estat molt positiva. És una manera nova d'aprendre, de conèixer gent que té els mateixos interessos que tu i et poden ensenyar moltes coses. Tot i que van ser dos dies molts intensos i, amb dies anteriors de preparació, tot l'esforç que es va fer ha estat útil per la realització i millora d'aquest treball i, sobretot per l'experiència d'haver participat en un esdeveniment així. Participant en aquesta trobada m'he adonat fins a quin punt Processing és una eina poderosa, cada vegada més utilitzada i amb un gran futur.

Al llarg d'aquests mesos de feina he trobat algunes dificultats esperades, per exemple, en la programació, i algunes de més inesperades, com el fet que el treball creatiu costa de formalitzar. El treball ha sigut molt variat:

- l'estudi de les lleis de la física, escriure les equacions en llenguatge de programació i comprovar que el resultat és l'esperat;
- la programació creativa;
- la participació a ProcessingBCN;
- la realització d'una peça final, que representa un petit projecte tancat, on s'ha d'aconseguir un resultat.

En resum, hem complert amb els objectius inicials del projecte, amb una feina que m'ha semblat molt interessant i diferent als treballs que s'han fet durant la carrera. Al tenir que utilitzar la creació ha donat lloc a una feina bastant divertida i entretinguda, tot i haver tingut moments difícils que requereixen el seu esforç.

BIBLIOGRAFIA

[1] Ben Fry, Casey Reas. Processing: A Programming Handbook for Visual Designers and Artists. MIT Press, Cambridge MA, 2007.

[2] Ira Greenberg. Processing: Creative Coding and Computational Art. Friends of Ed, Berkeley CA, 2007.

[3] Daniel Shiffman. Learning Processing: A Beginner's Guide to Programming Images, Animation and Interaction. Morgan Kaufmann, Burlington, MA, 2008.

[4] Paul A. Tipler, Gene Mosca. Física para la ciencia y la tecnología Vol. 1. Editorial Reverté 2005.

[5] Web "Ben Fry". http://benfry.com/

[6] Web "Casey Reas". http://reas.com/

[7] Web "coeficientes de fricción" pàgina web del projecte. http://www.monografias.com/trabajos15/coeficiente-friccion/coeficiente-friccion.shtml#FUERZA

[8] Web "col·lisions elàstiques". http://en.wikipedia.org/wiki/Elastic_collision

[9] Web "conservació del moment lineal". http://en.wikipedia.org/wiki/Momentum#Conservation_of_linear_momentum

[10] Web "foro de Processing en castellano visualP5". http://foro.visualp5.net/

[11] Web "llibreria Physics". http://www.cs.princeton.edu/~traer/physics/

[12] Web "Processing" pàgina web del projecte. http://www.processing.org/

[13] Web "Tutorial bàsic en francès de Processing". http://www.ecole-art-aix.fr/rubrique.php?id_rubrique=81

ANNEX A. CODI SKETCH

A.1. Classe Principal

```
//Importem les llibreries
import processing.opengl.*;
import traer.physics.*;
import ddf.minim.*;
import ddf.minim.analysis.*;

//Variables perla música
Minim minim;
AudioPlayer player;
BeatDetect beat;
BeatListener bl;

Dimension pantalla = Toolkit.getDefaultToolkit().getScreenSize();

int numParts = 281;  //nombre de partícules que s'aniran movent per la pantalla.

ParticleSystem din,est, sisFum;  // es tenen 3 sistemes: estàtic (les partícules invisibles no es mouen), dinàmic (partícules es mouen) i el fum.
Particle[] lletra = new Particle[150];  //Partícules que formaran la paraula ARU.
Particle[] parts = new Particle[numParts];  //Partícules que s'aniran movent per la pantalla.

int[] atretas = new int[numParts];  //Vector que guarda les partícules que es veuen afectades per l'atracció.
int[] agafades = new int[130];  // Vector que guarda quines partícules de la paraula ARU ja han creat atracció.
float[] tamany = new float[numParts];  //vector que guarda el tamany de les partícules.
float[] posInicialX = new float[numParts];  // vector que guarda la posició X de les partícules.
float[] posInicialY = new float[numParts];  // vector que guarda la posició Y de les partícules.

//Altres variables
int activat = 0, bimb=1, cont=0, fet=0, contador=0, atrac=0;
float angle=2*PI;


void setup()
{
  //Inicialització de la finestra.
  size( 900, 500, OPENGL );
  //Inicialització dels paràmetres del programa.
  frameRate( 40 );
  smooth();
  noStroke();

  //Carreguem la cançó i inicialitzem els paràmetres de la llibreria Minim.
  minim = new Minim(this);
  player = minim.loadFile("punk.mp3",2048);
  player.play();
  beat = new BeatDetect(player.bufferSize(), player.sampleRate());
  beat.setSensitivity(300);
  bl = new BeatListener(beat, player);

  //Inicialització dels tres sistemes.
  din = new ParticleSystem( 0, 0.1 );
  est = new ParticleSystem( 0, 0.1 );
  sisFum = new ParticleSystem( -0.15, 0.001 );

  creaEst();  //es creuen les partícules que formen la paraula ARU.
```

```
  //inicialitzem els vectors.
  for (int i=0; i<numParts; i++){
    atretas[i] = 0;
    tamany[i] = 1;
    if (i<130)
      agafades[i] = 0;
  }
}

void draw()
{
  frame.setLocation((pantalla.width-width)/2,(pantalla.height-height)/2);
  background(75,39,117);  //pintem el fons.

  //Fem avançar les simulacions dels sistemes.
  din.tick();
  sisFum.tick();

  //Es crida la funció de crear les partícules que es mouen fins que ja s'hagin creat totes.
  if (bimb< numParts){
    creaDin();
  }

  dibuixaDin(); //Es dibuixa les partícules.
  Movi();  // Es Mouen les partícules.
  ComprovaParets();  //Es comprova si les partícules han traspassat els límits de la finestra.

  //Es comprova si l'atracció ha estat activada.
  if (atrac==1){
    Atraccio();
    comprovaAtraccio();
  }

  //Es comprova si ja s'ha de crear el Fum.
  if (activat == 1){
    dibuixaFum();
  }

  //Amb el comptador es canvia el tipus de Moviment que escollim. El Moviment calcula les noves
posicions.
  if (contador == 460){
    Moviment2(200);
  }
  else if (contador == 620){
    Moviment3(200);
  }
  else if (contador== 800){
    Moviment4(200);
  }
  else if (contador== 940){
    Moviment5(200);

  }
  else if(contador==1000){
    atrac=1;
    contador = 400;
  }

  contador++;   //S'augmenta en 1 el comptador.

  //Quan s'hagin creat totes les partícules, es comença a interactuar amb la música.
  if(bimb > numParts-1)
  {
    Musica();
  }
}
```

```
void creaDin(){
  //Aquesta funció serveix per crear les partícules que es van movent. Es creen una en una, i en 7 grups
diferents.
  // Cada grup neix en un lloc diferent de la pantalla.
  if (bimb<(numParts-1)/7 + 1){
    parts[bimb] = din.makeParticle( 0.2, random(width*.5, width*.7), random(height*.8, height), 0 );
  }
  else if(bimb>(numParts-1)/7 && bimb< 2*(numParts-1)/7 + 1){
    parts[bimb] = din.makeParticle( 0.2, random(width*.7, width*.9), random(height*.4, height*.6), 0 );
  }
  else if (bimb>2*(numParts-1)/7 && bimb<3*(numParts-1)/7+1){
    parts[bimb] = din.makeParticle( 0.2, random(width*.2, width*.4), random(height*.1, height*.3), 0 );
  }
  else if(bimb>3*(numParts-1)/7 && bimb<4*(numParts-1)/7+1){
    parts[bimb] = din.makeParticle( 0.2, random(width*.8, width), random(height*.2), 0 );
  }
  else if (bimb>4*(numParts-1)/7 && bimb<5*(numParts-1)/7+1){
    parts[bimb] = din.makeParticle( 0.2, random(width*.2), random(height*.4, height*.6), 0 );
  }
  else if(bimb>5*(numParts-1)/7 && bimb<6*(numParts-1)/7 +1){
    parts[bimb] = din.makeParticle( 0.2, random(width*.4, width*.6), random(height*.4, height*.6), 0 );
  }
  else if(bimb>6*(numParts-1)/7 && bimb<numParts){
    parts[bimb] = din.makeParticle( 0.2, random(width*.2, width*.4), random(height*.7, height*.9), 0 );
  }
  // Es guarden les posicions inicials de cada partícula, per tal de poder crear el moviment.
  posInicialY[bimb] = parts[bimb].position().y();
  posInicialX[bimb] = parts[bimb].position().x();

  if (bimb<numParts){
    //Es crida el primer Moviment per calcular les noves posicions.
    Moviment1();
    bimb++;  //augmentem bimb per tal de passar a la pròxima partícula.
  }
}

void dibuixaDin(){
  //Aquesta funció serveix per pintar les partícules. Cada grup tindrà un color diferent, però tots dins la
tonalitat del blau cel.
  //Hi ha dos bucles, el primer pinta l'exterior de les partícules que tenen transperència. Després, es pinta
el centre.
  // A més, cada cop s'augmenta més el tamany fins arribar al màxim que és 12.
  for(int z = 1; z < bimb; z++){
    fill(2,234,229,80);
    if (z<(numParts-1)/7 + 1)  fill(137,237,235,80);
    if (z>(numParts-1)/7 && z<2*(numParts-1)/7 + 1)
      fill(97,201,198,80);
    if (z>2*(numParts-1)/7 && z<3*(numParts-1)/7 + 1)
      fill(31,252,247,80);
    if (z>3*(numParts-1)/7 && z<4*(numParts-1)/7 + 1)
      fill(114,188,186,80);
    if (z>4*(numParts-1)/7 && z<5*(numParts-1)/7 + 1)
      fill(65,188,188,80);
    if (z>5*(numParts-1)/7 && z<6*(numParts-1)/7 + 1)
      fill(73,250,232,80);
    ellipse(parts[z].position().x(), parts[z].position().y(), tamany[z], tamany[z]);
    if (tamany[z] <13){
      tamany[z] += 0.2;
    }
  }

  for(int z = 1; z < bimb; z++){
    fill(2,234,229);
    if (z<(numParts-1)/7 + 1)
      fill(137,237,235);
    if (z>(numParts-1)/7 && z<2*(numParts-1)/7 + 1)
```

```
      fill(97,201,198);
    if (z>2*(numParts-1)/7 && z<3*(numParts-1)/7 + 1)
      fill(31,252,247);
    if (z>3*(numParts-1)/7 && z<4*(numParts-1)/7 + 1)
      fill(114,188,186);
    if (z>4*(numParts-1)/7 && z<5*(numParts-1)/7 + 1)
      fill(65,188,188);
    if (z>5*(numParts-1)/7 && z<6*(numParts-1)/7 + 1)
      fill(73,250,232);
    ellipse(parts[z].position().x(), parts[z].position().y(), tamany[z]-6, tamany[z]-6);
    if (tamany[z] <13){
      tamany[z] += 0.2;
    }
  }
}

void ComprovaParets(){
  //Aquesta funció mira si les partícules sobrepassen el marge de la finestra. Si es així, passa directament
al costat oposat.

  for(int z = 1; z < bimb; z++){
    if(parts[z].position().y() > height){
      parts[z].moveTo( parts[z].position().x(), 0, 0 );  //Si la partícula ha passat del marge  dret, apareixerà
per l'esquerra.
    }
    else if(parts[z].position().y() < 0){
      parts[z].moveTo( parts[z].position().x(), height, 0 );  //Si la partícula ha passat del marge  esquerra,
apareixerà per la dreta.
    }
    else if(parts[z].position().x() > width){
      parts[z].moveTo(0, parts[z].position().y(), 0 );  //Si la partícula ha passat del marge  de baix, apareixerà
a dalt.
    }
    else if(parts[z].position().x() < 0){
      parts[z].moveTo( width, parts[z].position().y(), 0 );  //Si la partícula ha passat del marge  de dalt,
apareixerà baix.
    }
  }
}

void Atraccio(){
  //Aquesta funció serveix per crear l'atracció. S'ha dissenyat per a que cada partícula que forma la
paraula ARU només pugui atreure a una partícula en moviment.
  //Així algunes es seguiran movent, mentre els altres formen la paraula.

  for(int t = 1; t < 121; t++){ //recorrem les partícules de la paraula
    for (int v = 1; v <bimb; v++){  //recorrem les partícules en moviment.
      if(atretas[v] == 0){  //mirem si encara no té atracció
        if(agafades[t] < 1){  //mirem si la partícula de la paraula encara no ha creat atracció
          din.makeAttraction(lletra[t], parts[v] , 15000, 200);  //es crea l'atracció amb una força de 15000 i
distància de 200 a partir de la qual la força no augmenta.
          atretas[v] = 1; //es guarda la partícula que té atracció cap a la paraula.
          agafades[t]++;  // es guarda la partícula de la paraula ARU que ja ha creat l'atracció.
        }
      }
    }
  }
}

void dibuixaFum(){
 //Aquesta funció dibuixa el fum. Es recorren totes les partícules que té el sistema Fum i es pinta depenen
l'edat de cara partícula (així s'obté que cada cop sigui més transparent).
 //Si la partícula arriba a l'edat de 40, es destrueix.
  for ( int i = 0; i < sisFum.numberOfParticles(); i++)
  {
    Particle p = sisFum.getParticle( i );
    fill(97,201,198, 255/(p.age()+1) );
```

```
    ellipse( p.position().x(), p.position().y(), 7, 7 );
    if ( p.age() > 40 )
      p.kill();
  }
}

void creaFum(int a){
  //Aquesta funció crea les partícules de Fum.
  //Se li afegeix velocitat per tal de que vagi per amunt.
  Particle p = sisFum.makeParticle( 1, parts[a].position().x(), parts[a].position().y(), 0 );
  p.setVelocity( random( -1, 1 ), random( -.6, -1.5 ), 0 );
  activat = 1;  //es posa a 1 el paràmetre que fa que es dibuixi el fum.
}

void comprovaAtraccio(){
  //Aquesta funció serveix per comprovar si les partícules ja estan a la posició de la paraula.
  // Es suposa que estan ben col·locades quan la distància entre la partícula que forma la paraula ARU i la
que és atreta és menor a 7. Si es així, es crida la funció de crear el Fum.
  float d=0.0;
  for (int v=1; v<bimb; v++){
    if (atretas[v] != 0){
      for (int t=1; t<121; t++){
        d = dist (lletra[t].position().x(), lletra[t].position().y(), parts[v].position().x(), parts[v].position().y());
        if (d<7){
          creaFum(v);
        }
      }
    }
  }
}

void Movi(){
  //Aquesta funció serveix per moure les partícules.
  //Primer es mira si totes les partícules del mateix grup han estat creades, ja que fins que no estan totes
creades no es comença a moure el grup.

  if(tamany[(numParts-1)/7]>12){
    cont = (numParts-1)/7 +1;
  }
  if(tamany[2*(numParts-1)/7]>12){
    cont = 2*(numParts-1)/7 +1;
  }
  if(tamany[3*(numParts-1)/7]>12){
    cont = 3*(numParts-1)/7 +1;
  }
  if(tamany[4*(numParts-1)/7]>12){
    cont = 4*(numParts-1)/7 +1;
  }
  if(tamany[5*(numParts-1)/7]>12){
    cont = 5*(numParts-1)/7 +1;
  }
  if(tamany[6*(numParts-1)/7]>12){
    cont = 6*(numParts-1)/7 +1;
  }
  if(tamany[7*(numParts-1)/7]>12){
    cont = 7*(numParts-1)/7 +1;
  }

  for(int i=1;i<cont;i++){
    if (atrac==1 && atretas[i]==0){  //es mira si ja s'ha creat l'atracció i es multiplica la velocitat per 5, ja que
l'atracció relantilitza les partícules.
      parts[i].moveBy(5*incrementX[i], 5*incrementY[i],0);  // Les variables incrementX i incrementY es
calculen a la classe Moviment
    }
    else{  //si no s'ha creat l'atracció, la partícula es mou depenent de la velocitat calculada.
      parts[i].moveBy(incrementX[i], incrementY[i],0);
    }
```

```
  }
}

void Musica(){
  //Aquesta funció serveix per interacturar amb la música.
  for (int i=0; i<bimb; i++){ //Es mira si el tamany de les partícules es més gran de 12, si es així es redueix
lentament.
    if (tamany[i]>12){
      tamany[i] *= 0.9;
    }
  }

  //Es mira si s'ha detectat alguns d'aquets tons: Kick, Snare o Hat. Si es així, s'agafen dues partícules
aleatories i s'augmenta el tamany.
  if ( beat.isKick()) {
    tamany[int(random(1,numParts))]=25;
    tamany[int(random(1,numParts))]=25;
  }
  if ( beat.isSnare() ){
    tamany[int(random(1,numParts))]=35;
    tamany[int(random(1,numParts))]=35;
  }
  if ( beat.isHat() ){
    tamany[int(random(1,numParts))]=45;
    tamany[int(random(1,numParts))]=45;
  }
}

void init(){
  frame.dispose();
  frame.setUndecorated(true);
  super.init();
}

void stop()
{
  //Aquesta funció serveix per aturar la cançò i tancar la llibria Minim.
  player.close();
  minim.stop();
  super.stop();
}
```

A.2. Classe ARU

```
void creaEst(){
   //lletra A.
  lletra[1] = est.makeParticle(6, 0.188*width, 0.740*height,0);
  lletra[1].makeFixed();
  lletra[2] = est.makeParticle(6, 0.195*width, 0.710*height,0);
  lletra[2].makeFixed();
  lletra[3] = est.makeParticle(6, 0.202*width, 0.680*height,0);
  lletra[3].makeFixed();
  lletra[4] = est.makeParticle(6, 0.208*width, 0.650*height,0);
  lletra[4].makeFixed();
  lletra[5] = est.makeParticle(6, 0.215*width, 0.620*height,0);
  lletra[5].makeFixed();
  lletra[6] = est.makeParticle(6, 0.222*width, 0.590*height,0);
  lletra[6].makeFixed();
  lletra[7] = est.makeParticle(6, 0.228*width, 0.560*height,0);
  lletra[7].makeFixed();
  lletra[8] = est.makeParticle(6, 0.233*width, 0.530*height,0);
  lletra[8].makeFixed();
  lletra[9] = est.makeParticle(6, 0.240*width, 0.500*height,0);
```

```
lletra[9].makeFixed();
lletra[10] = est.makeParticle(6, 0.246*width, 0.470*height,0);
lletra[10].makeFixed();
lletra[11] = est.makeParticle(6, 0.253*width, 0.440*height,0);
lletra[11].makeFixed();
lletra[12] = est.makeParticle(6, 0.260*width, 0.410*height,0);
lletra[12].makeFixed();
lletra[13] = est.makeParticle(6, 0.266*width, 0.380*height,0);
lletra[13].makeFixed();
lletra[14] = est.makeParticle(6, 0.273*width, 0.350*height,0);
lletra[14].makeFixed();
lletra[15] = est.makeParticle(6, 0.280*width,0.320*height,0);
lletra[15].makeFixed();
lletra[16] = est.makeParticle(6, 0.286*width, 0.290*height,0);
lletra[16].makeFixed();
lletra[17] = est.makeParticle(6, 0.293*width, 0.260*height,0);
lletra[17].makeFixed();
lletra[18] = est.makeParticle(6, 0.297*width,0.290*height,0);
lletra[18].makeFixed();
lletra[19] = est.makeParticle(6, 0.302*width,0.320*height,0);
lletra[19].makeFixed();
lletra[20] = est.makeParticle(6, 0.306*width, 0.350*height,0);
lletra[20].makeFixed();
lletra[21] = est.makeParticle(6, 0.311*width, 0.380*height,0);
lletra[21].makeFixed();
lletra[22] = est.makeParticle(6, 0.315*width, 0.410*height,0);
lletra[22].makeFixed();
lletra[23] = est.makeParticle(6, 0.320*width, 0.440*height,0);
lletra[23].makeFixed();
lletra[24] = est.makeParticle(6, 0.324*width, 0.470*height,0);
lletra[24].makeFixed();
lletra[25] = est.makeParticle(6, 0.328*width, 0.500*height,0);
lletra[25].makeFixed();
lletra[26] = est.makeParticle(6, 0.333*width, 0.530*height,0);
lletra[26].makeFixed();
lletra[27] = est.makeParticle(6, 0.337*width, 0.560*height,0);
lletra[27].makeFixed();
lletra[28] = est.makeParticle(6, 0.342*width, 0.590*height,0);
lletra[28].makeFixed();
lletra[29] = est.makeParticle(6, 0.346*width, 0.620*height,0);
lletra[29].makeFixed();
lletra[30] = est.makeParticle(6, 0.351*width, 0.650*height,0);
lletra[30].makeFixed();
lletra[31] = est.makeParticle(6, 0.355*width, 0.680*height,0);
lletra[31].makeFixed();
lletra[32] = est.makeParticle(6, 0.360*width,  0.710*height,0);
lletra[32].makeFixed();
lletra[33] = est.makeParticle(6, 0.364*width, 0.740*height,0);
lletra[33].makeFixed();
lletra[34] = est.makeParticle(6, 0.222*width, 0.600*height,0);
lletra[34].makeFixed();
lletra[35] = est.makeParticle(6, 0.235*width, 0.600*height,0);
lletra[35].makeFixed();
lletra[36] = est.makeParticle(6, 0.248*width, 0.600*height,0);
lletra[36].makeFixed();
lletra[37] = est.makeParticle(6, 0.262*width, 0.600*height,0);
lletra[37].makeFixed();
lletra[38] = est.makeParticle(6, 0.275*width, 0.600*height,0);
lletra[38].makeFixed();
lletra[39] = est.makeParticle(6, 0.288*width, 0.600*height,0);
lletra[39].makeFixed();
lletra[40] = est.makeParticle(6, 0.302*width, 0.600*height,0);
lletra[40].makeFixed();
lletra[41] = est.makeParticle(6, 0.315*width, 0.600*height,0);
lletra[41].makeFixed();
lletra[42] = est.makeParticle(6, 0.328*width, 0.600*height,0);
lletra[42].makeFixed();
```

```
lletra[43] = est.makeParticle(6, 0.342*width, 0.600*height,0);
lletra[43].makeFixed();

//lletra R.
lletra[44] = est.makeParticle(6, 0.444*width,0.746*height,0);
lletra[44].makeFixed();
lletra[45] = est.makeParticle(6, 0.444*width,0.716*height,0);
lletra[45].makeFixed();
lletra[46] = est.makeParticle(6, 0.444*width,0.686*height,0);
lletra[46].makeFixed();
lletra[47] = est.makeParticle(6, 0.444*width,0.656*height,0);
lletra[47].makeFixed();
lletra[48] = est.makeParticle(6, 0.444*width,0.626*height,0);
lletra[48].makeFixed();
lletra[49] = est.makeParticle(6, 0.444*width,0.596*height,0);
lletra[49].makeFixed();
lletra[50] = est.makeParticle(6, 0.444*width,0.566*height,0);
lletra[50].makeFixed();
lletra[51] = est.makeParticle(6, 0.444*width,0.536*height,0);
lletra[51].makeFixed();
lletra[52] = est.makeParticle(6, 0.444*width,0.506*height,0);
lletra[52].makeFixed();
lletra[53] = est.makeParticle(6, 0.444*width,0.476*height,0);
lletra[53].makeFixed();
lletra[54] = est.makeParticle(6, 0.444*width,0.446*height,0);
lletra[54].makeFixed();
lletra[55] = est.makeParticle(6, 0.444*width,0.416*height,0);
lletra[55].makeFixed();
lletra[56] = est.makeParticle(6, 0.444*width,0.386*height,0);
lletra[56].makeFixed();
lletra[57] = est.makeParticle(6, 0.444*width,0.356*height,0);
lletra[57].makeFixed();
lletra[58] = est.makeParticle(6, 0.444*width,0.326*height,0);
lletra[58].makeFixed();
lletra[59] = est.makeParticle(6, 0.444*width,0.296*height,0);
lletra[59].makeFixed();
lletra[60] = est.makeParticle(6, 0.444*width,0.266*height,0);
lletra[60].makeFixed();
lletra[61] = est.makeParticle(6, 0.464*width,0.270*height,0);
lletra[61].makeFixed();
lletra[62] = est.makeParticle(6, 0.484*width,0.276*height,0);
lletra[62].makeFixed();
lletra[63] = est.makeParticle(6, 0.511*width,0.276*height,0);
lletra[63].makeFixed();
lletra[64] = est.makeParticle(6, 0.525*width,0.294*height,0);
lletra[64].makeFixed();
lletra[65] = est.makeParticle(6, 0.542*width,0.322*height,0);
lletra[65].makeFixed();
lletra[66] = est.makeParticle(6, 0.548*width,0.362*height,0);
lletra[66].makeFixed();
lletra[67] = est.makeParticle(6, 0.552*width,0.398*height,0);
lletra[67].makeFixed();
lletra[68] = est.makeParticle(6, 0.544*width,0.432*height,0);
lletra[68].makeFixed();
lletra[69] = est.makeParticle(6, 0.533*width,0.464*height,0);
lletra[69].makeFixed();
lletra[70] = est.makeParticle(6, 0.517*width,0.486*height,0);
lletra[70].makeFixed();
lletra[71] = est.makeParticle(6, 0.498*width,0.498*height,0);
lletra[71].makeFixed();
lletra[72] = est.makeParticle(6, 0.481*width,0.496*height,0);
lletra[72].makeFixed();
lletra[73] = est.makeParticle(6, 0.461*width,0.490*height,0);
lletra[73].makeFixed();
lletra[74] = est.makeParticle(6, 0.457*width,0.520*height,0);
lletra[74].makeFixed();
lletra[75] = est.makeParticle(6, 0.474*width,0.518*height,0);
```

```
lletra[75].makeFixed();
lletra[76] = est.makeParticle(6, 0.496*width,0.524*height,0);
lletra[76].makeFixed();
lletra[77] = est.makeParticle(6, 0.512*width,0.536*height,0);
lletra[77].makeFixed();
lletra[78] = est.makeParticle(6, 0.525*width,0.556*height,0);
lletra[78].makeFixed();
lletra[79] = est.makeParticle(6, 0.530*width,0.586*height,0);
lletra[79].makeFixed();
lletra[80] = est.makeParticle(6, 0.531*width,0.628*height,0);
lletra[80].makeFixed();
lletra[81] = est.makeParticle(6, 0.536*width,0.676*height,0);
lletra[81].makeFixed();
lletra[82] = est.makeParticle(6, 0.541*width,0.710*height,0);
lletra[82].makeFixed();
lletra[83] = est.makeParticle(6, 0.548*width,0.730*height,0);
lletra[83].makeFixed();

//lletra U.
lletra[84] = est.makeParticle(6, 0.633*width, 0.266*height,0);
lletra[84].makeFixed();
lletra[85] = est.makeParticle(6, 0.633*width, 0.296*height,0);
lletra[85].makeFixed();
lletra[86] = est.makeParticle(6, 0.633*width, 0.326*height,0);
lletra[86].makeFixed();
lletra[87] = est.makeParticle(6, 0.633*width, 0.356*height,0);
lletra[87].makeFixed();
lletra[88] = est.makeParticle(6, 0.633*width, 0.386*height,0);
lletra[88].makeFixed();
lletra[89] = est.makeParticle(6, 0.633*width, 0.416*height,0);
lletra[89].makeFixed();
lletra[90] = est.makeParticle(6, 0.633*width, 0.446*height,0);
lletra[90].makeFixed();
lletra[91] = est.makeParticle(6, 0.633*width, 0.476*height,0);
lletra[91].makeFixed();
lletra[92] = est.makeParticle(6, 0.633*width, 0.506*height,0);
lletra[92].makeFixed();
lletra[93] = est.makeParticle(6, 0.633*width, 0.536*height,0);
lletra[93].makeFixed();
lletra[94] = est.makeParticle(6, 0.633*width, 0.566*height,0);
lletra[94].makeFixed();
lletra[95] = est.makeParticle(6, 0.633*width, 0.596*height,0);
lletra[95].makeFixed();
lletra[96] = est.makeParticle(6, 0.633*width, 0.626*height,0);
lletra[96].makeFixed();
lletra[97] = est.makeParticle(6, 0.633*width, 0.656*height,0);
lletra[97].makeFixed();
lletra[98] = est.makeParticle(6, 0.644*width, 0.686*height,0);
lletra[98].makeFixed();
lletra[99] = est.makeParticle(6, 0.653*width, 0.716*height,0);
lletra[99].makeFixed();
lletra[100] = est.makeParticle(6, 0.670*width, 0.740*height,0);
lletra[100].makeFixed();
lletra[101] = est.makeParticle(6, 0.686*width, 0.752*height,0);
lletra[101].makeFixed();
lletra[102] = est.makeParticle(6, 0.707*width, 0.752*height,0);
lletra[102].makeFixed();
lletra[103] = est.makeParticle(6, 0.725*width, 0.752*height,0);
lletra[103].makeFixed();
lletra[104] = est.makeParticle(6, 0.747*width, 0.738*height,0);
lletra[104].makeFixed();
lletra[105] = est.makeParticle(6, 0.763*width, 0.720*height,0);
lletra[105].makeFixed();
lletra[106] = est.makeParticle(6, 0.777*width, 0.694*height,0);
lletra[106].makeFixed();
lletra[107] = est.makeParticle(6, 0.786*width, 0.670*height,0);
lletra[107].makeFixed();
```

```
  lletra[108] = est.makeParticle(6, 0.786*width, 0.640*height,0);
  lletra[108].makeFixed();
  lletra[109] = est.makeParticle(6, 0.786*width, 0.610*height,0);
  lletra[109].makeFixed();
  lletra[110] = est.makeParticle(6, 0.786*width,0.580*height,0);
  lletra[110].makeFixed();
  lletra[111] = est.makeParticle(6, 0.786*width,0.550*height,0);
  lletra[111].makeFixed();
  lletra[112] = est.makeParticle(6, 0.786*width,0.520*height,0);
  lletra[112].makeFixed();
  lletra[113] = est.makeParticle(6, 0.786*width,0.490*height,0);
  lletra[113].makeFixed();
  lletra[114] = est.makeParticle(6, 0.786*width,0.460*height,0);
  lletra[114].makeFixed();
  lletra[115] = est.makeParticle(6, 0.786*width,0.430*height,0);
  lletra[115].makeFixed();
  lletra[116] = est.makeParticle(6, 0.786*width,0.400*height,0);
  lletra[116].makeFixed();
  lletra[117] = est.makeParticle(6, 0.786*width,0.370*height,0);
  lletra[117].makeFixed();
  lletra[118] = est.makeParticle(6, 0.786*width,0.340*height,0);
  lletra[118].makeFixed();
  lletra[119] = est.makeParticle(6, 0.786*width,0.310*height,0);
  lletra[119].makeFixed();
  lletra[120] = est.makeParticle(6, 0.786*width,0.280*height,0);
  lletra[120].makeFixed();
}
```

A.3. Classe BeatListener

```
class BeatListener implements AudioListener
{
  private BeatDetect beat;
  private AudioPlayer source;

  BeatListener(BeatDetect beat, AudioPlayer source)
  {
    this.source = source;
    this.source.addListener(this);
    this.beat = beat;
  }

  void samples(float[] samps)
  {
    beat.detect(source.mix);
  }

  void samples(float[] sampsL, float[] sampsR)
  {
    beat.detect(source.mix);
  }
}
```

A.4. Classe Moviment

```
float[] incrementX = new float[numParts];  //Vector que guarda els increments de la posicio X (seria com la velocitat en l'eix X).
float[] incrementY = new float[numParts];  //Vector que guarda els increments de la posicio Y (seria com la velocitat en l'eix Y).
int[] mov = new int[numParts];  // Vector que guarda en quin moviment està la partícula. Aixo serveix
```

perque no totes faran el mateix moviment alhora.

```
void Moviment1(){
  //Primer moviment --> Cap a l'eix (0,0)
  //Es calcula els increments que hauria de fer la partícula per arribar a la posició (0,0). Sempre amb un
poc de aleatorietat.
  incrementX[bimb] = abs(cos(angle*random(0.2)))*(20 - posInicialX[bimb]) / 500;
  incrementY[bimb] = (20 - posInicialY[bimb]) / random(400,500);
  mov[bimb] = 0;  //Es guarda 0 a les partícules afectades per aquest moviment.
}

void Moviment2(int num){
  //Segon moviment --> Cap a l'eix (0,height/2)
  int i=0, j= 0;
  while(j<num){
    i = int(random(1,numParts));  //S'agafen num partícules perque segueixen aquesta nova trajectòria.
    if (mov[i] != 1){
      mov[i] = 1; //Es guarda 1 a les partícules afectades per aquest moviment.
      incrementX[i] = abs(cos(angle*random(0.2)))*(20 - parts[i].position().x()) /700;
      incrementY[i] = ((height - 20) - parts[i].position().y()) / random(500,600);
      j++;
    }
  }
}

void Moviment3(int num){
  //Tercer moviment --> Cap a l'eix (0.55*width,height)
  int i=0, j= 0;
  while(j<num){
    i = int(random(1,numParts));  //S'agafen num partícules perque segueixen aquesta nova trajectòria.
    if (mov[i] < 2){
      mov[i] = 2;  //Es guarda 2 a les partícules afectades per aquest moviment.
      incrementX[i] = abs(cos(angle*random(0.1)))*(0.55*width - parts[i].position().x()) /350; //random
(250,300);
      incrementY[i] = ((height - 20) - parts[i].position().y()) / random(200,300); //random (250,300);
      j++;
    }
  }
}

void Moviment4(int num){
  //Quart moviment --> Cap a l'eix (width,0)
  int i=0, j= 0;
  while(j<num){
    i = int(random(1,numParts));  //S'agafen num partícules perque segueixen aquesta nova trajectòria.
    if (mov[i] < 3){
      mov[i] = 3;  //Es guarda 3 a les partícules afectades per aquest moviment.
      incrementX[i] = abs(cos(angle*random(0.2)))*(width - parts[i].position().x()) /250; //random (250,300);
      incrementY[i] = (20 - parts[i].position().y()) / random(200,300); //random (250,300);
      j++;
    }
  }
}

void Moviment5(int num){
  //Igual que el primer moviment --> Cap a l'eix (0,0)
  int i=0, j= 0;
  while(j<num){
    i = int(random(1,numParts));  //S'agafen num partícules perque segueixen aquesta nova trajectòria.
    if (mov[i] < 4){
      mov[i] = 4;  //Es guarda 4 a les partícules afectades per aquest moviment.
      incrementX[i] = abs(cos(angle*random(0.2)))*(20 - parts[i].position().x()) /150; //random (250,300);
      incrementY[i] = (40 - parts[i].position().y()) / random(300,400); //random (250,300);
      j++;
    }
  }
}
```

ANNEX B. PRESENTACIÓ PROCESINGBCN

Introducción a la programación de movimiento y los sistemas de partículas

Joana Sansaloni

Escola Politècnica Superior de Castelldefels
Universitat Politècnica de Catalunya

ProcessingBCN - Mayo 2009

Processing. Primitivas básicas

Parámetros del dibujo

```
size(w,h);
background(color);
stroke(color);
strokeWeight(g);
noStroke();
fill(color);
noFill();
```

Variables del sistema

```
width
height
mouseX
mouseY
pmouseX
pmouseY
```

Colores

```
blanco y negro := entero entre 0 i 255
blanco y negro, canal alfa := 2 enteros
RGB := 3 enteros
RGB, con canal alfa := 4 enteros
```

Primitivas de dibujo

```
point(x,y);
line(x1,y1,x2,y2);
rect(x,y,w,h);
ellipse(x,y,w,h);
```

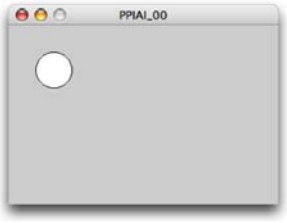
ProcessingBCN - Mayo 2009

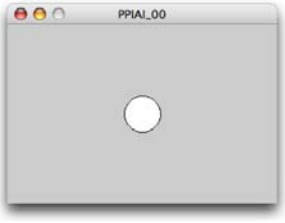
Processing. Coordenadas

Atención:
(0,0) es el vértice
superior izquierdo

ellipse(50,50,40,40);

size(100,100);
ellipse(50,50,40,40);





size(300,200);
ellipse(50,50,40,40);

size(300,200);
ellipse(width/2,height/2,40,40);

ProcessingBCN - Mayo 2009

Processing. Animación

Actualizar la matriz de píxeles correspondiente al dibujo.
Volver a pintar.

Processing: modo continuo de programación.

```
void setup(){
  inicializaciones del dibujo
}
void draw(){
  bucle infinito: se ejecuta un cierto # de veces por segundo
}
```

ProcessingBCN - Mayo 2009

Processing. Objeto en movimiento

En cada frame se realizan las acciones:

```
calcular_posicion();
dibujar();
mover();
```

Con distintas opciones:

```
background(color);
```
el objeto se dibuja cada vez.

```
fill(color,alfa);
rect(0,0,width,height)
```
el objeto deja una estela.

se va dibujando toda la trayectoria.

ProcessingBCN - Mayo 2009

Processing. Objeto en movimiento

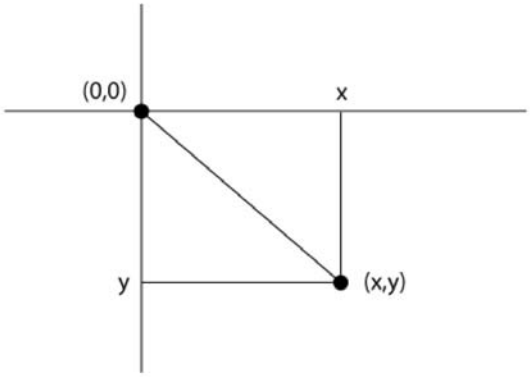
Maneras de mover un objeto: dar la nueva posición

- Interacción
- Seguir una trayectoria
  Uso de matemáticas. Curvas.
- Sistema con fuerzas actuando
  Uso de física. Más de un objeto.
  Movimientos simples: uniforme, uniformemente acelerado, circular, armónico.
  Tiro parabólico.
  Librería Physics. Colisiones.

En cualquier caso, se pueden incluir elementos aleatorios con la función `random()`

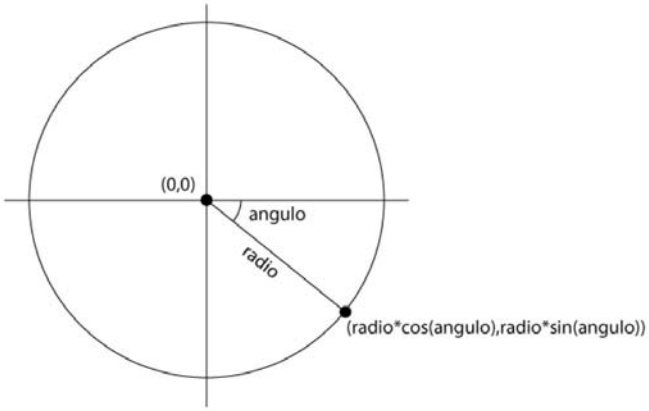
ProcessingBCN - Mayo 2009

Processing. Coordenadas cartesianas



ProcessingBCN - Mayo 2009

Processing. Coordenadas polares

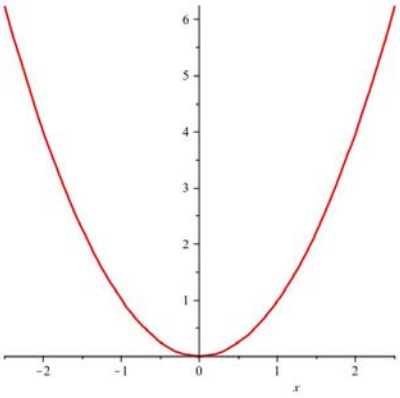


ProcessingBCN - Mayo 2009

Processing. Trayectorias

Matemáticas

$$y = x^2$$



Processing

```
void setup(){
  background(0);
  fill(255);
  stroke(255);

  i=0;
}

void draw(){
  background(0);
  x=i;
  y=i*i;
  ellipse(x,y,5,5);

  i++;
}
```

ProcessingBCN - Mayo 2009

Processing. Movimiento simple

Física: uniforme

$$x = x_0 + v \cdot t$$

Processing

```
void setup(){
  x=10; // x_0
  v=2;  // v
}

void draw(){
  ellipse(x,0,5,5);
  x+=v;
}
```

```
void setup(){
  x=10; // x_0
  y=5;
  vX=2;  // v
  vY=1;
}

void draw(){
  ellipse(x,y,5,5);
  x+=vX;
  y+=vY;
}
```

ProcessingBCN - Mayo 2009

Processing. Movimiento simple

Física: uniformemente acelerado

$$x = x_0 + v_0 \cdot t + a * t^2$$

Processing

```
void setup(){
  x=10; // x_0
  v=2;  // v_0
  a=1; // a
}
void draw(){
  ellipse(x,0,5,5);
  v+=a;
  x+=v;
}
```

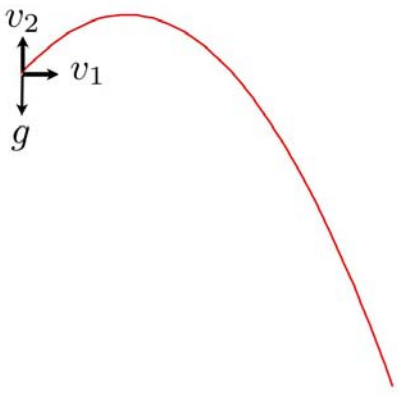
ProcessingBCN - Mayo 2009

Processing. Movimiento simple

Física: tiro parabólico

$$x = x_0 + v_1 * t$$

$$y = y_0 + v_2 * t + g * t^2$$



Processing

```
void setup(){
  x=50; // x_0
  y=20;
  v1=2;  // v_1
  v2=-2;
  g=0.5; // g
}
void draw(){
  ellipse(x,y,5,5);
  x+=v1;
  v2+=g;
  y+=v2;
}
```

ProcessingBCN - Mayo 2009

Processing. Movimiento simple

Física: circular

$$x = r * cos(\omega * t)$$
$$y = r * sin(\omega * t)$$

Processing

```
void setup(){
  r=100; // radio
  angulo=0; // t inicial
  w=0.02;  // v. angular
}

void draw(){
  translate(width/2,height/2);
  x=r*cos(angulo);
  y=r*sin(angulo);
  ellipse(x,y,5,5);
  angulo+=w;
}
```

ProcessingBCN - Mayo 2009

Processing. Movimiento simple

Física: armónico

$$x = A * sin(\omega * t + \theta)$$

Processing

```
void setup(){
  A=100; // amplitud
  i=0; // t
  w=0.02;  // v. angular
  f=1; // fase
}

void draw(){
  translate(width/2,height/2);
  angulo=w*i+f;
  x=A*sin(angulo);
  ellipse(x,0,5,5);
  i++;
}
```

ProcessingBCN - Mayo 2009

Enlaces

Curvas y más

http://local.wasp.uwa.edu.au/~pbourke/
http://mathworld.wolfram.com/
http://www.2dcurves.com/

Librería Physics

http://www.cs.princeton.edu/%7Etraer/physics/

Colisiones

http://en.wikipedia.org/wiki/Momentum
http://en.wikipedia.org/wiki/Elastic_collision

Ejemplos

http://visualp5.net/processingbcn/pg/file/lali/read/118/ejemplos-movimiento-y-partculas

ProcessingBCN - Mayo 2009

ANNEX C. INSTAL·LACIÓ DEL PROGRAMA

Instal·lar Processing a l'ordinador és molt senzill. Primerament s'ha de descarregar el software des de http://processionalment/download/. Està disponible per Windows, Linux i Mac OSX.

Si s'utilitza Windows hi ha l'opció de baixar-se'l sense Java per anar més ràpid, però necessita una instal·lació prèvia de la versió correcta de Java. És més fàcil baixar-se la versió normal, encara que tardi una mica més.

Una vegada baixat el programa només s'ha de descomprimir i obrir l'aplicació. Automàticament, es crea una carpeta a “Mis Documentos” anomenada Processing on es guarden els sketches per defecte i les llibreries.

EL programa ja porta incorporades algunes de les llibreries, però per a la resta s'han de descarregar per separat i s'han de col·locar dins de la carpeta “Libraries” que està dins la carpeta Processing.